5-31-1990

Multichannel input moniter

Geng Lin Chen
New Jersey Institute of Technology

ABSTRACT

Title of Thesis: Multichannel input monitor

Geng Lin Chen, Master of Science, 1990

Thesis directed by:
Dale T. Teaney Ph.D.
Associate Professor
Department of Electrical Engineering

In order to communicate with each other, people must use the same kind of language. MIDI, the Musical Instrument Digital Interface, was established as a hardware and software specification which makes the communication between musical instruments become possible.

What this thesis does is interfacing a musical instrument with other devices using MIDI protocol. The hardware is a multichannel priority interrupt system which has already been done and was included in the discussion of this thesis. The focus of this thesis is concentrated on the software development of multichannel input monitor.

MULTICHANNEL INPUT MONITOR

by
Geng Lin Chen

Thesis submitted to the Faculty of the Graduate School of the New Jersey Institute of Technology in partial fulfillment of the requirements for the degree of
Master of Science
1990

APPROVAL SHEET

Title of Thesis: Multichannel input monitor

Name of Candidate: Geng Lin Chen
Master of Science, 1990

Thesis and Abstract Approved: ______________________ 3/12/90
Dale Teaney Ph.D. Date
Associate Professor
Department of Electrical Engineering

______________________ 3/28/90
Date

Signatures of other members
of the thesis committee.

______________________ 3/28/90
Date

VITA

Name: Geng Lin Chen

Permanent address:

Degree and date to be conferred: Master of Science, 1990

Date of birth:

Place of birth:

Secondary education: Provincial Hsin-Chu Senior High School, 1979

Collegiate institutions attended	Dates	Degree	Date of Degree
Chinese Culture University Yang-Ming-Shan, Taiwan, R.O.C.	Sep. 1979 to July 1980		
Tamkang University Tamsui, Taiwan 25137, R.O.C.	Sep. 1980 to July 1983	Bachelor of Science	June 1983
New Jersey Institute of Technology Newark, New Jersey 07102	Sep. 1988 to May 1990	Master of Science	May 1990

Major: Electrical Engineering

TABLE OF CONTENTS

Chapter		Page
	INTRODUCTION	1
I.	HARDWARE DESCRIPTION	2
	A. Principle of Hardware	2
	B. I/O Configuration	9
	C. Set Up MIDIVIBS	10
II.	SOFTWARE DESCRIPTION	12
	A. System Flow	16
	B. Handling of Argument Arrays	17
	C. Programs Detail	20
	D. Function of Programs	28
	E. Memory Map of Programs	30
	F. Flowcharts	34
III.	TEST	40
	A. Testing Program	40
	B. Flowcharts of Testing Program	42
	C. Hardware Setup for Test	44
	D. Analysis of Test Result	46
	1. Example 1	46
	2. Example 2	48
	3. Example 3	50
APPENDIX A	INTRODUCTION TO MIDI	55
APPENDIX B	PROGRAMS LIST	60
APPENDIX C	ARCHITECTURE OF SBC	90
REFERENCES		99

LIST OF FIGURES

Figure		Page
1.1	Simplified system block diagram of MIDIVIBS	4
1.2	Cards connection of MIDIVIBS.	5
1.3	Diagram to set up MIDIVIBS.	11
2.1	The cylinder mechanism.	14
2.2	Note status transition diagram for INT1 interrupt routine.	26
2.3	Note status transition diagram for main program . . .	26
2.4	The 8052 Program Memory	32
2.5	The 8052 Data Memory.	32
2.6	External 64K bytes memory map of program MIDIVIBS . .	33
3.1	Diagram of test system setup.	44
3.2	The circuit of MIDI Data Collection Interface	45
3.3	A simple example MIDI file.	46
3.4	Time diagram of MIDI file in Fig 3.3.	46
3.5	MIDI file for example2.	48
3.6	Time diagram of MIDI file in Fig 3.5.	49
3.7	MIDI file for example 3	50
3.8	Time diagram of MIDI file in Fig 3.7.	54
4.1	The different MIDI messages	58
4.2	The different controllers under MIDI Control Change message	59

LIST OF TABLES

Table | Page

2.1 5 possible Note statuses are represented by high nibble of Status byte 23

2.2 Transition of Note status in INT1 interrupt routine . 25

2.3 Transition of Note status in main program 27

LIST OF FLOWCHARTS

Flowchart		Page
1	Arrays Handling Algorithm Part I (main program). . . .	18
2	Arrays Handling Algorithm Part II (INT1 service routine)	19
3	Main Program	34
4	Renew Note Array Subroutine.	35
5	Processing Note Array One Cycle Subroutine	36
6	INT1 Interrupt Routine	37
7	INT1 Note Array Handling Subroutine.	38
8	TIMER0 Interrupt Routine	
	(1 MS Timer For MIDIVIBS program).	39
9	MIDI Data Collection Program	42
10	TIMER0 Interrupt Routine	
	(1 MS Timer For MIDI Data Collection program). . . .	43

INTRODUCTION

The purpose of this thesis is to build an interface between vibraphone and other musical instrument using MIDI protocol. The input of this interface device is the analog signal produced by sensors in the vibraphone. The output of this interface device is MIDI protocol digital signal. For convenience we call this interface device MIDIVIBS.

The hardware of MIDIVIBS which has been done by Claudio Bernal and Chen-Tung Mo includes 4 different cards. They are Analog I/O card, Key Card, Interface card and SBC card. Chapter one describes the principle of hardware and functions of these cards [1,2].

Chapter two is the software algorithm and detail software description. This chapter introduces a cylinder mechanism from which the main algorithm of software comes out. It is the research result of Dr. Teaney and Binghong Gui.

Chapter three discusses the test of MIDIVIBS. This chapter includes the hardware diagram and software algorithm for MIDIVIBS test. It also includes the result of test.

Appendix A is an Introduction to MIDI protocol [3].

The programs which are used by this thesis are included in Appendix B.

Appendix C is the Architecture of SBC. This SBC is designed by Dr. Teaney.

CHAPTER ONE

HARDWARE DESCRIPTION

A. Principle of Hardware

Fig 1.1 is the simplified system block diagram of the MIDIVIBS. Following are the description of how the system works.

The input analog signals are picked up by vibration sensors. The typical wave form of input signal is shown on the block diagram in Fig 1.1. This analog signal is fed to a positive active filter circuit which rectifies it to be a positive envelope as shown in Fig 1.1, then this positive signal is fed to low pass filter circuit. The output of this low pass filter is also shown in Fig 1.1. The next block circuit is Peak Hold which can detect and hold the peak of the input signal. When the input signal arrives its maximum this circuit will output a peak velocity to the Multiplexer circuit. At the same time this Peak Hold circuit will send an INT signal to Cascaded priority encoder circuit which can make SBC responses to the interrupt events according to the priority of channels.

When the program in SBC receives INT 1 signal it will read the output of Cascaded priority encoder (at location 1111 0xxx xxxx x000 B) which tells the highest priority channel currently requests the interrupt service. Then the program sends out the ID of this channel to the Multiplexer circuit, so the velocity of this channel can pass through to the Analog multiplier and A/D converter circuits. Then the program reads the velocity from A/D converter and does proper processing of this channel. In order to enable interrupt signal from other channels this channel must be reset. To reset this channel, first, the program sends out this channel's ID to Cascaded demultiplexer which enables the reset pass from SBC to this channel. Then the program sends out a reset signal. This reset signal will remove the INT signal of this channel away, so the INT signal of lower priority channel can be processed.

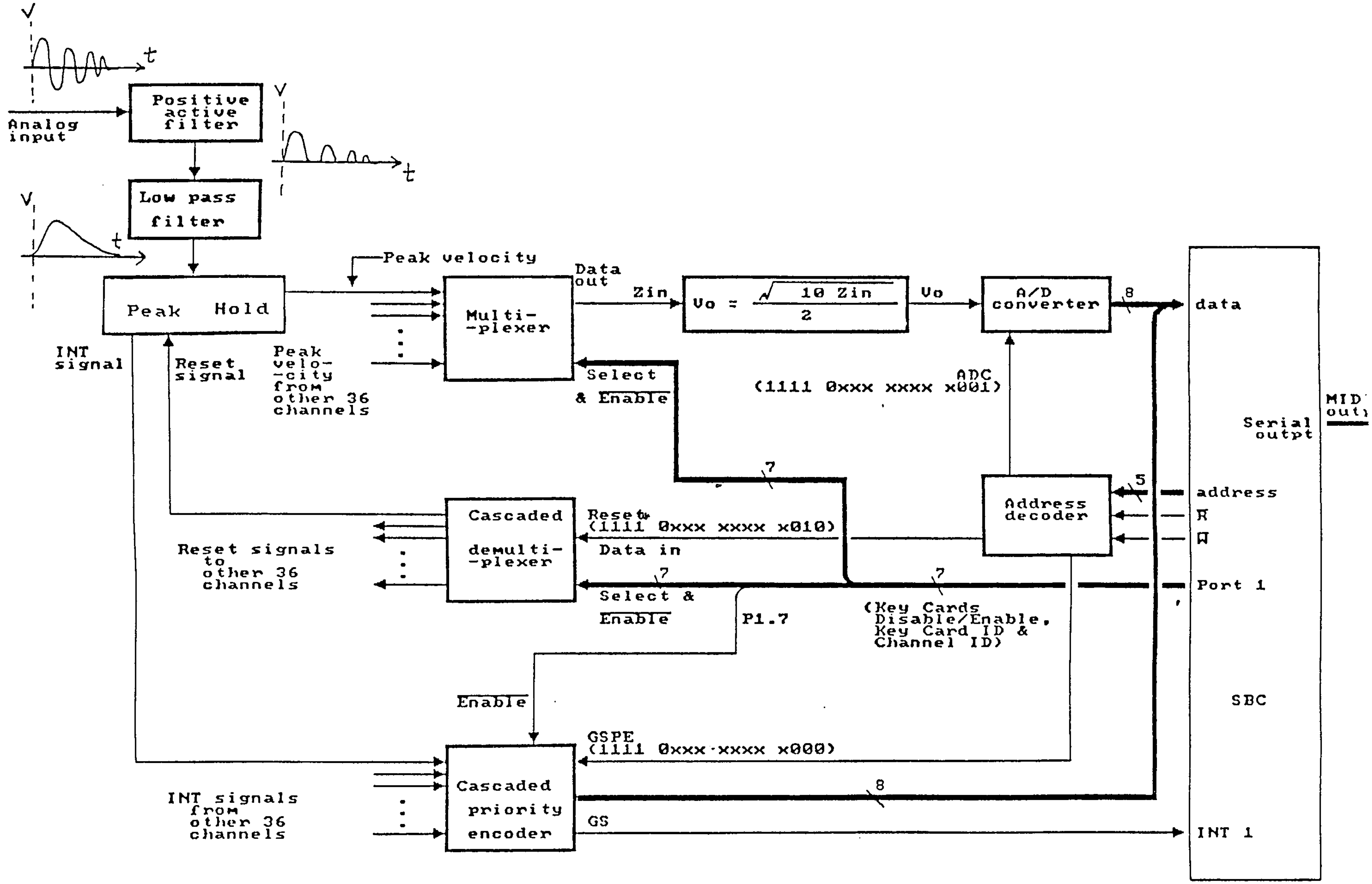


Fig 1.1 Simplified system block diagram of MIDIVIBS

Fig 1.2 is the cards connection of MIDIVIBS. There are 4 different cards. The source signals are connected to the Analog I/O Card. The output of Analog I/O Card is connected to 5 Key Cards. Then these 5 Key Cards are connected to Interface Card. The Interface Card then connected to SBC card from which the MIDI signal comes out. Following paragraphs are functions of these cards.

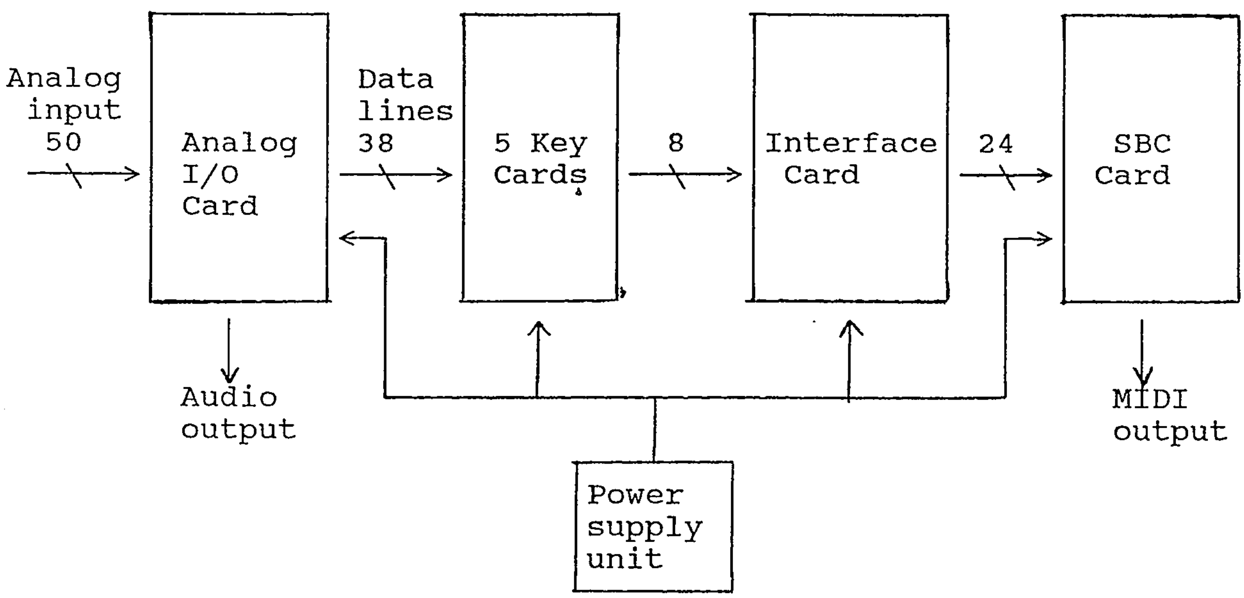


Fig 1.2 Cards connection of MIDIVIBS

Analog I/O Card

The source signals of MIDIVIBS are picked up by 37 vibration transducers which are stuck on every keys on the vibraphone. These vibration signals are connected to Analog I/O Card by two 25 lines parallel cables. This Analog I/O card provides 37 channels driving current to next boards (5 Key Cards). Also there is an Audio Output in this card from which we can get analog audio signal of the vibraphone [2].

Key Card

There are 5 Key Cards connect to Analog I/O Card. The circuit of these 5 Key Cards are all the same. Each Key Card handles 8 signal channels. Because there are only 37 channels input signal two channels of the first Key card and one channel of the fifth Key card are not been used.

The 8 channels in Key card are controlled by Analog Multiplexer (MC14051). This decoder decides whether one of the 8 channels is enabled or all channels are disabled. For each channel there is Peak Hold circuit which will detect and hold maximum amplitude of input analog signal in this channel. This maximum amplitude is called the velocity of the KEY. The velocity will be held until program reset this channel. This Peak Hold also creates a signal which is connected to priority encoder (MC14532). The priority encoder of 5 Key cards are chained, so the higher priority encoder can disable the lower priority encoder [2].

Interface Card

The 5 Key Cards are connected to Interface Card. This card provides level shifter to transfer data from CMOS voltage level to TTL voltage level and from TTL voltage level to CMOS voltage level. In this card there is a second priority decoder. The inputs of this priority decoder are connected to the GS pin of priority encoder of 5 Key Cards. From the GS pin of this second cascaded priority encoder the SBC's interrupt pin is connected (INT1) . These priority encoders decide the priority of channels [1].

The Interface Card also provide analog to digit converter. The input signal of ADC is provided by following formula:

$$Va = \frac{\sqrt{10 * Velocity}}{2}$$

Where Va is the analog input of ADC and Velocity is the velocity output connected to 5 Key Cards. The square root function is achieved by Analog multiplier (ICL 8013).

The address decoder is also in this card which provides the memory map I/O.

SBC Card

The SBC Card which includes one 8052AH-BASIC microprocessor is an independent computer board. It includes ROM and RAM which are basic elements for a program to be executed. The program is burned in an 8K EPROM. The serial output of microprocessor is connected to MIDI OUT where the MIDI message is sent out. Appendix C is the architecture of SBC.

B. I/O Configuration

Except using Port 1 as a direct I/O , the hardware also uses memory mapped I/O. This memory mapped I/O uses partial decoding technique. Following is the binary address of memory mapped I/O and their function. Where x means don't care.

Memory Mapped I/O

BINARY ADDRESS	R/W	FUNCTION	DATA
1111 0xxx xxxx x000	R	Read GSPE (Group Selected Priority Encoder) which tells the highest priority channel currently active.	GSPE (explained bellow)
1111 0xxx xxxx x001	W	Reset A/D converter (Start to convert)	Don't Care
1111 0xxx xxxx x001	R	Read A/D converter	0-127
1111 0xxx xxxx x010	W	Reset the channel which is currently selected by the data at Port 1.	Don't Care

GSPE: bit 7 and bit 3 are always zero.

bit 4 to bit 6 represent the Key Card ID.

	Bit 6	Bit 5	Bit 4
Key Card 1	0	1	1
Key Card 2	1	0	0
Key Card 3	1	0	1
Key Card 4	1	1	0
Key Card 5	1	1	1

bit 0 to bit 2 represent the Key Channel ID.

	Bit 2	Bit 1	Bit 0
Key Channel 1	0	0	0
Key Channel 2	0	0	1
Key Channel 3	0	1	0
Key Channel 4	0	1	1
Key Channel 5	1	0	0
Key Channel 6	1	0	1
Key Channel 7	1	1	0
Key Channel 8	1	1	1

Direct I/O

Port 1: all 8 bits of Port 1 are used as output.

bit 7 Disable/Enable Key Card.

When this bit is high the 5 Key Cards are disabled.

When it is low the 5 Key Cards are Enabled.

bit 3 In order to execute program this bit must be set to high always.

bit 0 to bit 2 and bit 4 to bit 6 are the same definitions as in the GSPE. These 6 bits decide which Key Channel at which Key Card is enabled. Bit 7 of Port 1 must be low in order to enable the specified Key Card.

C. Set Up MIDIVIBS

To set up MIDIVIBS, it is simply by connecting vibraphone to the MIDIVIBS and then connecting the MIDI Output to other MIDI devices. For example, we can connect the MIDI output with a computer which has the MIDI facility to record the song playing on the vibraphone, or connect the MIDI output with any musical instruments using MIDI protocol. Fig 1.3 is an example of set up MIDIVIBS.

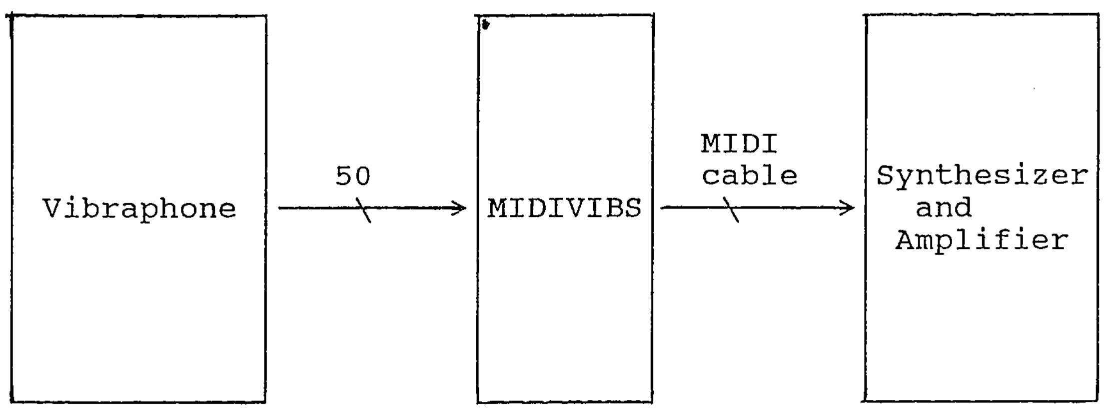


Fig 1.3 Diagram to set up MIDIVIBS

CHAPTER TWO

SOFTWARE DESCRIPTION

Before start to describe the software algorithm and programs detail, one important mechanism from which the main algorithm of the software comes out must be described first. This mechanism is worked out by Dr. Teaney and is shown in Fig 2.1.

Fig 2.1(a) is an empty cylinder which represents the situation before any element is active. There are 37 element sources. When any element is active, it will be pushed into the cylinder. Following are the functions of the cylinder components.

The Input Gate on the top of the cylinder has two positions, open and close. When there are new elements above the Input Gate and the Test & Eject part is not executing its testing cycle, the new elements can be pushed into the cylinder. When the Test & Eject part is executing its testing cycle, this Input Gate is locked at close position. The Input Gate in Fig 2.1(a) is at its close position. Fig 2.1(b) shows how 3 elements are pushed into the cylinder. the Input Gate at the left cylinder of Fig 2.1(b) is at its open position.

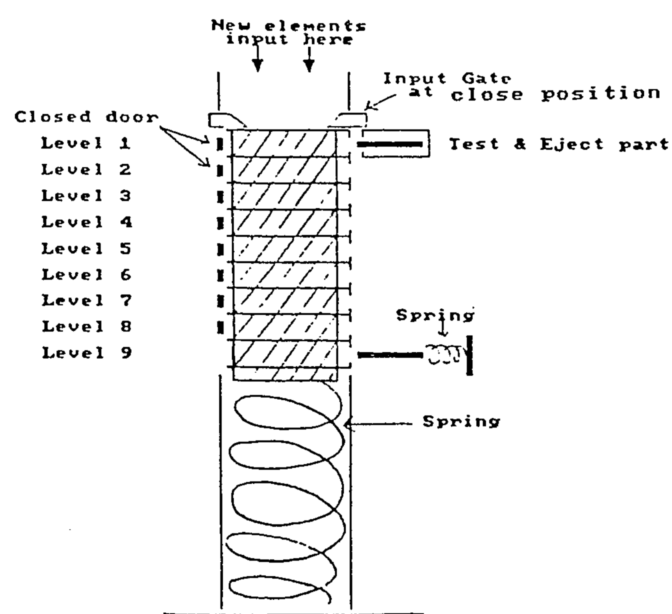
The Test & Eject part on the right side of the cylinder can detect the activity of the element. It will compare the activity, Va of this element with the threshold, Vth. If Va is less then Vth it will eject this element from the left side of the cylinder. This Test & Eject part will examine every elements in the cylinder from level 1 to level N where N is the number of elements current in the cylinder. Fig 2.1(c) shows 4 elements in the cylinder while the Test & Eject part is executing its testing cycle. The Test & Eject part always starts a new testing cycle from level 1. Before it starts a new testing cycle it will unlock the Input Gate so that the new elements above the Input Gate can be pushed into the cylinder. After all new elements above Input Gate have been pushed into the cylinder, it locks the Input Gate at its close position and begins testing cycle. So, before the end of testing cycle, no new element can be pushed into the cylinder. Each time when Test & Eject part ejects one element instead of move to next level as the condition Va > Vth, the Test & Eject part will stay at the same level and test next element. When there is no more element below the level where the Test & Eject part is located, the Test & Eject part will return to level 1 and end this testing cycle.

On the left side of the cylinder, there are eight doors. When the Test & Eject part is not at its testing cycle these 8 doors are closed. When it is at its testing cycle only one of these eight doors is opened which is the one at the same level as the Test & Eject part. For example, during the testing cycle the Test & Eject part is at level 3 then the door at level 3 is opened. So, when the activity of this element is less then threshold the Test & Eject part can eject it.

The spring at level 9 on the right side of the cylinder will eject automatically the element which is pushed onto level 9. That means when there are 9 elements under the Input Gate the element at level 9 will be ejected automatically. This keeps the maximum number of elements inside the cylinder to be eight.

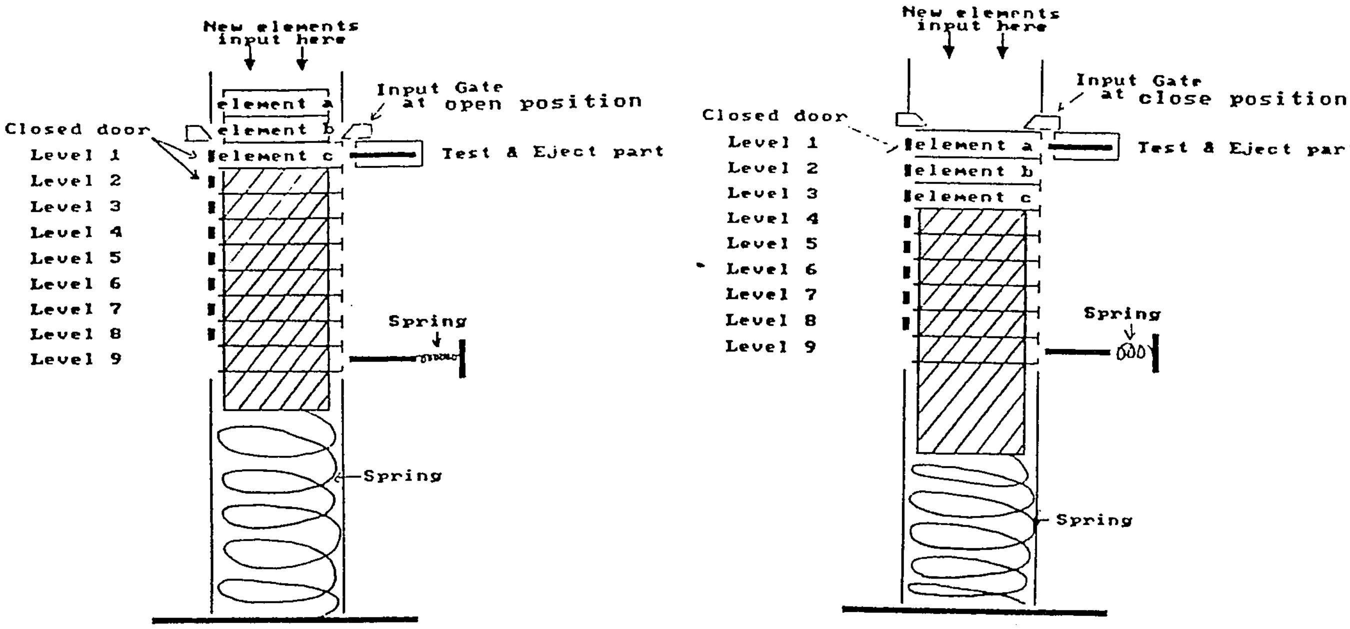
The spring under the cylinder makes sure that there is no empty place inside the cylinder. Each time when the Test & Eject part ejects one element out, the spring will push the elements under the element which was ejected one level up.

The reason to implement this mechanism into software algorithm is to improve the efficiency of software. The array handling algorithm which is described on next two sections is the implementation of this mechanism. The 37 Keys on the vibraphone represent the elements of this mechanism. The arrays described on next two sections represent the cylinder. The velocity of note represents the activity of the element.

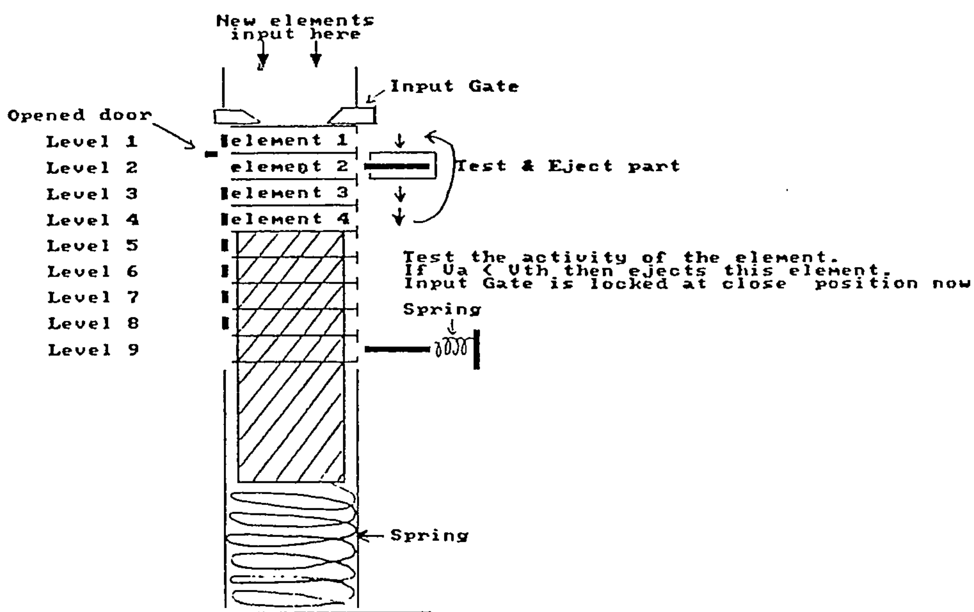


(a) Empty cylinder

Fig 2.1 The cylinder mechanism



(b) 3 elements are pushed into the cylinder



(c) Test & Eject part at its testing cycle

Fig 2.1 The cylinder mechanism

A. System Flow

The algorithm of software can be described roughly as following. This algorithm and next section "Handling of Argument Arrays" are research results Dr. Teaney and Binghong Gui.

a) arguments used

M(8) -- note array of maximum size 8.

M(1) is the newest data.

IO() -- new data array of maximum size 8.

IO(I) is the newest data, "I" is the pointer to the newest data.

b) The main program keeps checking the note array M(). If the velocity of one note is lower than the threshold, then the note will be turned off.

c) If an interrupt from vibraphone is coming, the computer will go to serve the INT routine. The INT routine collects the new data in the array IO(8), with IO(I) is the newest data. The IO() only keeps the newest 8 data.

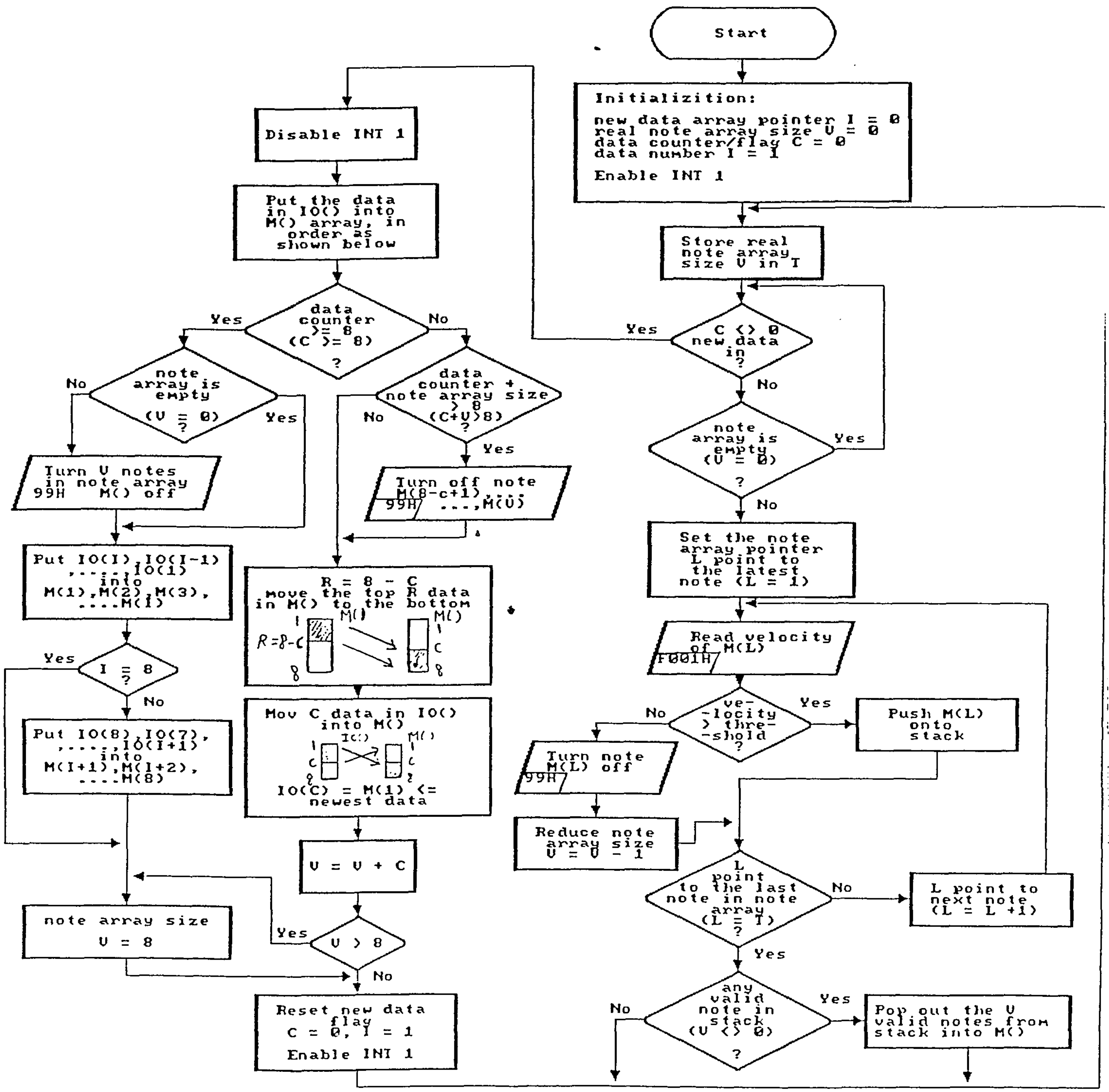
d) After coming back from INT routine, the note array M() need to be reorganized with new data IO(). Then the control goes to the main note checking routine (step b).

B. Handling of Argument Arrays

The argument arrays discussed here are M(8) and IO(8). As describes previously M(8) is note array of maximum size 8 and M(1) is the newest data. IO(8) is new data array of maximum size 8 where IO(I) is the newest data.

Flowchart 1 and 2 on page 18 and 19 are array handling algorithm that shows the algorithm of handling these two arrays. Note that when the main program checks the note array M() it uses stack as a storage place for valid notes. This simplifies the arrangement of the order of elements in M() and makes the program more efficient.

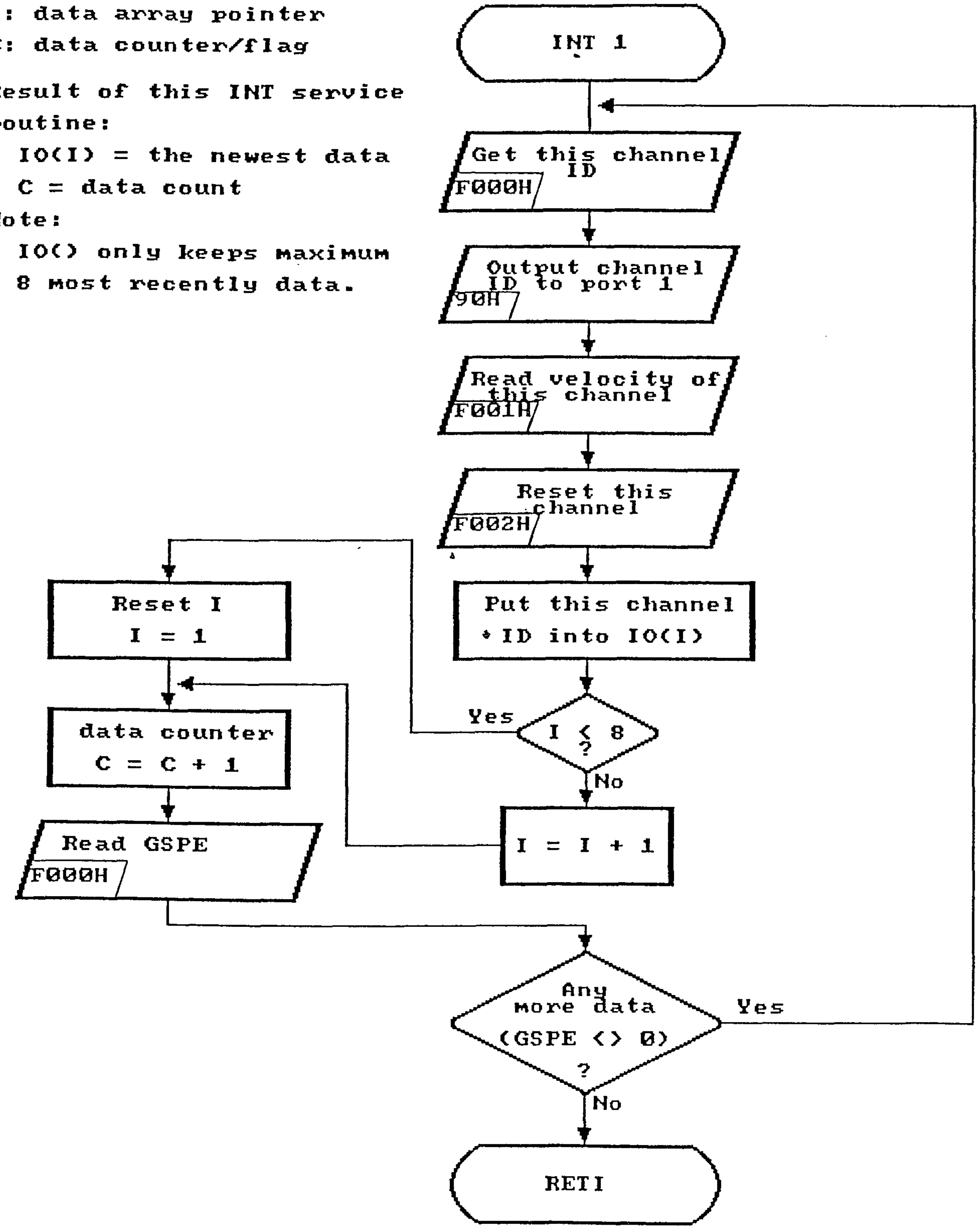
Flowchart 1 Arrays Handling Algorithm Part I (main program)



Flowchart 2 Arrays Handling Algorithm Part II
(INT 1 service routine)

I: data array pointer
C: data counter/flag

Result of this INT service routine:
  IO(I) = the newest data
  C = data count
Note:
  IO() only keeps maximum 8 most recently data.



C. Programs Detail

In the beginning the program is written by BASIC. Because the speed of BASIC is too slow, it is necessary to transfer the BASIC into assembly which is the first program in the Appendix B. This first program called MIDIVIBS will be discussed in this chapter. The second program in the Appendix B which is written by BASIC and assembly is a program that can monitor the MIDI DATA output. It will be discussed in next chapter.

The flowcharts of MIDIVIBS are from page 34 to 39. The argument array handling algorithm is the same as shown on Flowchart 1 and Flowchart 2. The line number beside each block in the flowchart is the corresponding program line number which executes the function described in that block. The address on the left down corner of I/O block is the location where this I/O action is took place.

The MIDI message used by this thesis is only the Note On command. The Note OFF command is replaced by a Note ON command with zero attack velocity. Also the program use the running status technique to speed up the processing. When the program is sending turn note on MIDI message it sends status byte 90H. When it is sending turn note off message no status byte will be sent out. For more information about MIDI protocol see Appendix A, Introduction to MIDI [3].

The software basically includes 6 programs. In order to make this program more efficient it uses 2 bytes space for each Note on the vibraphone. Because of these two extra bytes the program doesn't need to waste its time in delay loop. One of these two extra bytes is STATUS byte. The other is TM byte. The TM byte is used to store time at which this note changes its status. The STATUS byte is to store Note status. The status byte is explained bellow.

Bit 7 of status byte which is not used is always zero. Bit 0 to Bit 3 of status byte are status time count value. If this low nibble is not zero then its value will be decreased by one every 10 ms. Not until the low nibble of status byte is zero this note can not change its status which is represented by high nibble of status byte.

Bit 4 to Bit 6 of status byte represent 5 situations. Following are the meaning of these three bits.

Bit 4 of status byte: If this note has just been turned ON no more than 50 ms or has just been turned OFF no more than 30 ms then this bit will be set. Otherwise it will be cleared.

Bit 5 of status byte: If this bit is set that means this note is ON right now. If this bit is cleared that means this note is OFF right now.

Bit 6 of status byte: If this bit is set then this note will be turned ON 30 ms later with the velocity read at that time. The reason to use this status bit is because it is not proper to turn this note ON when this note has just been turned OFF no more than 30 ms.

For example, this note has just been turned ON no more than 50 ms if you turn it OFF immediately your output MIDI message will be a note ON message with a duration less than 50 ms. This is a too short duration for you to hear this note.

On the other hand if this note has just been turned OFF no more than 30 ms and you turn it ON immediately you will get a MIDI note OFF message with a duration less than 30 ms.

So actually bit 6 of status byte is used to make sure there is enough Note OFF and Note ON time.

For example, if status byte is 35H. The high nibble of this status byte is 3 and the low nibble of status byte is 5. That means this note is in status 4 (this note has just been turned ON) and not until 50 ms later the program will not check the velocity of this note.

Although there are 8 different combinations for bit 4 to bit 6. There are only 5 possible statuses which are used in this program. These 5 statuses are shown in Table 2.1.

Note Status	Status byte bit 6	Status byte bit 5	Status byte bit 4
1	0	0	0
2	0	0	1
3	0	1	0
4	0	1	1
5	1	0	0

Table 2.1: 5 possible Note statuses are represented by high nibble of Status byte

Note: bit 7 : always zero

bit 6: 30 ms later read velocity flag

bit 5: note ON flag

bit 4: If this bit is set that means right now it is 50 ms time period after note ON or 30 ms time period after note OFF.

Table 2.2 is the statuses transition table and Fig 2.2 is the statuses transition diagram for INT1 interrupt routine. Table 2.3 is the statuses transition table and Fig 2.3 is the statuses transition diagram for main program.

Following are the explanation of 5 possible Note statuses.

Status 1: (status byte high nibble = 0) This note has been turned OFF for more than 30 ms.

Status 2: (status byte high nibble = 1) This note has just been turned OFF no longer than 30 ms ago.

Status 3: (status byte high nibble = 2) This note has been turned ON more than 50 ms and it is ON right now.

Status 4: (status byte high nibble = 3) This note has just been turned ON no longer than 50 ms ago.

Status 5: (status byte high nibble = 4) This note will be turned ON no more than 30 ms later.

Note Status before processing	Transition of Note status

1	Check velocity
	IF velocity >= threshold THEN
	BEGIN
	turn note ON;
	Note status 1 -> Note status 4;
	store current time into TM byte;
	END
	ELSE
	BEGIN
	Note status 1 -> Note status 5;
	store current time into TM byte;
	END;

2	Note status 2 -> Note status 5; store current time into TM byte;
3	Note status 3 -> Note status 5; turn note OFF; store current time into TM byte;
4 or 5	no change of status byte;

Table 2.2 Transition of Note status in INT1 interrupt routine.

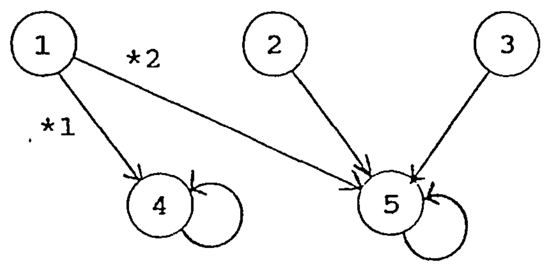


Fig 2.2 Note status transition diagram for INT1 interrupt routine.

note: *1 velocity >= threshold

*2 velocity < threshold

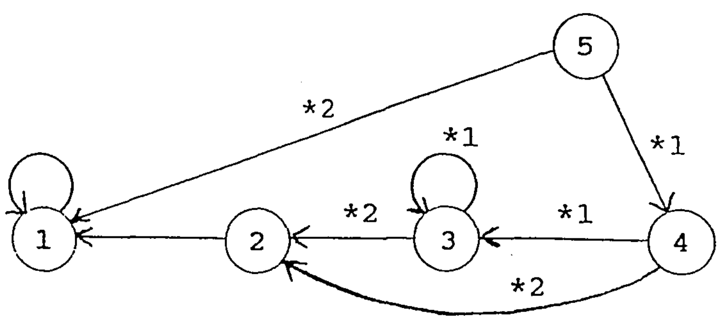


Fig 2.3 Note status transition diagram for main program.

note: *1 velocity >= threshold

*2 velocity < threshold

Note Status before processing	Transition of Note status
1	no processing of Note status
2	Note status 2 -> Note status 1; store current time into TM byte;

3 or 4	Check velocity
	IF velocity < threshold THEN
	  BEGIN
	    turn note OFF;
	    Note status 3 or 4 -> Note status 2;
	    store current time into TM byte;
	  END
	ELSE
	  BEGIN
	    Note status 3 or 4 -> Note status 3;
	    store current time into TM byte;
	  END;

Table 2.3 Transition of Note status in main program.

(This table is continued on next page)

Note Status before processing	Transition of Note status

5	Check velocity
	
	IF velocity < threshold THEN
	BEGIN
	
	Note status 5 -> Note status 1
	
	store current time into TM byte;
	
	END
	
	ELSE
	
	BEGIN
	
	Note status 5 -> Note status 4
	
	turn note ON with velocity gets from A/D converter;
	
	store current time into TM byte;
	
	END;

Table 2.3 Transition of Note status in main program.

(Continued from previous page)

D. FUNCTION OF PROGRAMS

Following are functions of 6 assembly programs of MIDIVIBS:

Program one "Main Program" (flowchart 3) : Main routine includes initialization and control of the main flow.

Program two "Renew Note Array Subroutine" (flowchart 4) : This program is called by the main program. It will add new notes which are currently gotten by INT 1 interrupt routine and are currently in the Interrupt Note Array IO(). If the total Note number includes in the Current Note Array M() and Interrupt Note Array IO() are more than 8 then this program will keep the newest 8 notes and turn others off.

Program three "Processing Current Note Array One Cycle Subroutine" (flowchart 5) : This program is called by the main program. It will check current velocity for those notes in Current Note Array M(). If the velocity of one note is less than threshold then it will turn this note OFF and move it out from Current Note Array. Notice that if the note has just been checked no more than 10 ms ago then this program will not check this note in this processing cycle. Not only 10 ms has to be passed from last time checking of this Note, but also the low nibble of status byte of this note has to be zero, otherwise the program will not check the velocity of this note. It will only decrease the low nibble of the status byte of this note by 1.

That means the value of 10 ms count of status byte of this Note is decreased by one. The meaning of status byte is explained in section C, Programs Detail, in this chapter.

Program four "INT 1 Interrupt Routine" (flowchart 6) : This program will be executed when there is a note on vibraphone be hit. It will assign different status to this note according to current status of this Note. The transition of status byte is explained in Table 2.2 and Fig 2.2.

Program five "INT1 Interrupt Note Array Handling Subroutine" (flowchart 7) : This program is called by INT 1 interrupt routine. It will put current Note into Interrupt Note Array, This routine also keeps no more than 8 newest motes.

Program six "TIMER0 Interrupt Routine" (flowchart 8) : What this interrupt routine does is just increasing the 1 ms counter by one and load timer0 registers with proper value in order to create 1 ms timer (For the crystal of system clock equals 11 MHz, the value is FC6CH).

E. MEMORY MAP OF PROGRAM

Fig 2.4 and Fig 2.5 are the memory maps of MCS8052-BASIC [5].

In the BASIC system there must be at least 1K external RAM located at the bottom (low address) of external 64K bytes memory. Also MCS8052-BASIC mirrors the interrupt vectors to external memory location 4000H. So if the program includes interrupt service routines under BASIC system, there must be memory device (ROM or RAM) at location 4000H to put the interrupt vector.

In order to use the auto boot feature of MCS8052-BASIC under BASIC system, there must be ROM begins at location 8000H. The minimum size of this ROM area depends on the size of the BASIC program to be executed. For example, the BASIC program is 100H bytes than the minimum size of this ROM will be 100H plus 11H which is used to store system informations. That comes to 111H bytes.

So if the program is executed under system of BASIC the minimum number of memory ICs will be 3 for SBC board. One begins at 0. The other begins at 4000H. Another begins at 8000H.

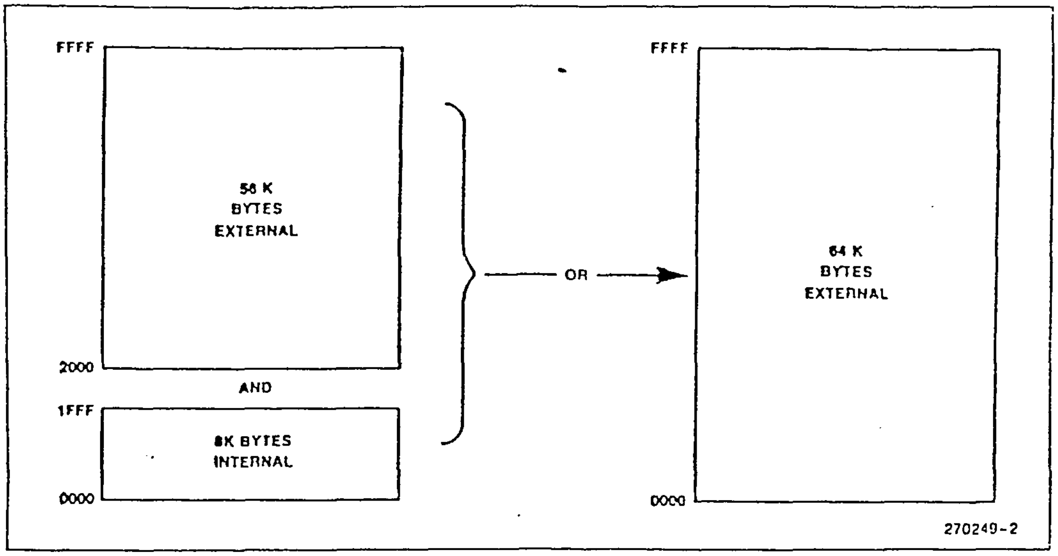


Fig 2.4 The 8052 Program Memory

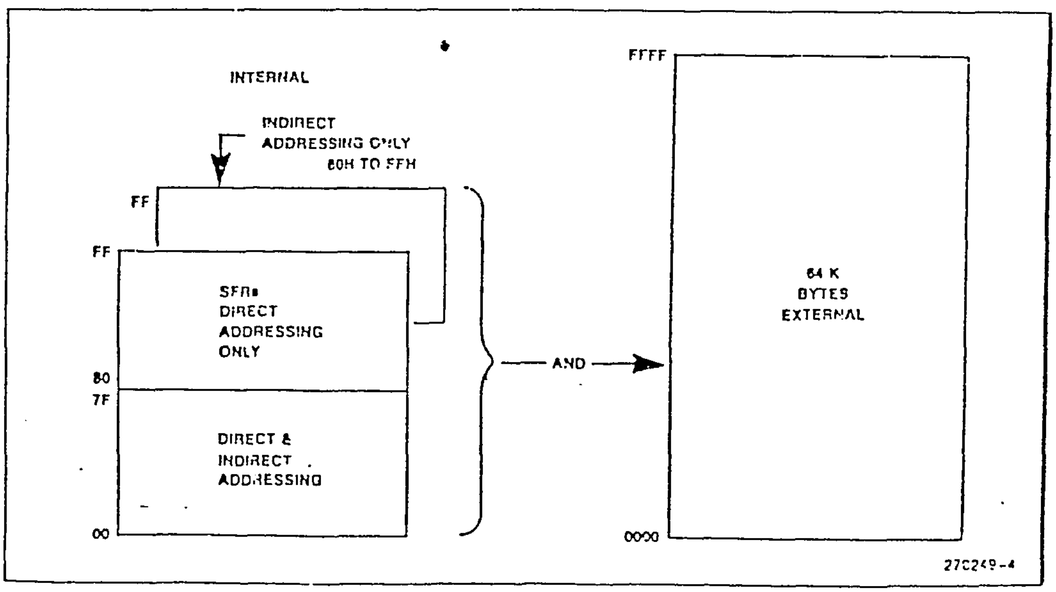


Fig 2.5 The 8052 Data Memory

The program in this thesis has already been transferred into assembly. It is not necessary to run this program under BASIC system. Because the program. in this thesis uses the external reset feature of MCS8052-BASIC, so it only needs 2 memory ICs. One is for the code area starts at 2000H. The other is for the data area starts at 4000H. Fig 2.6 is the memory map of the program MIDIVIBS.

0H - 1FFFH	NOT USED
2000H - 25FFH	CODE AREA (ROM)
2600H - 3FFFH	NOTE USED
4000H - 43FFH	DATA AREA (RAM)
4400H - DFFFH	NOT USED
E000H - FFFFH	I/O AREA

Fig 2.6 External 64K bytes memory map of program MIDIVIBS.

D. FLOWCHARTS

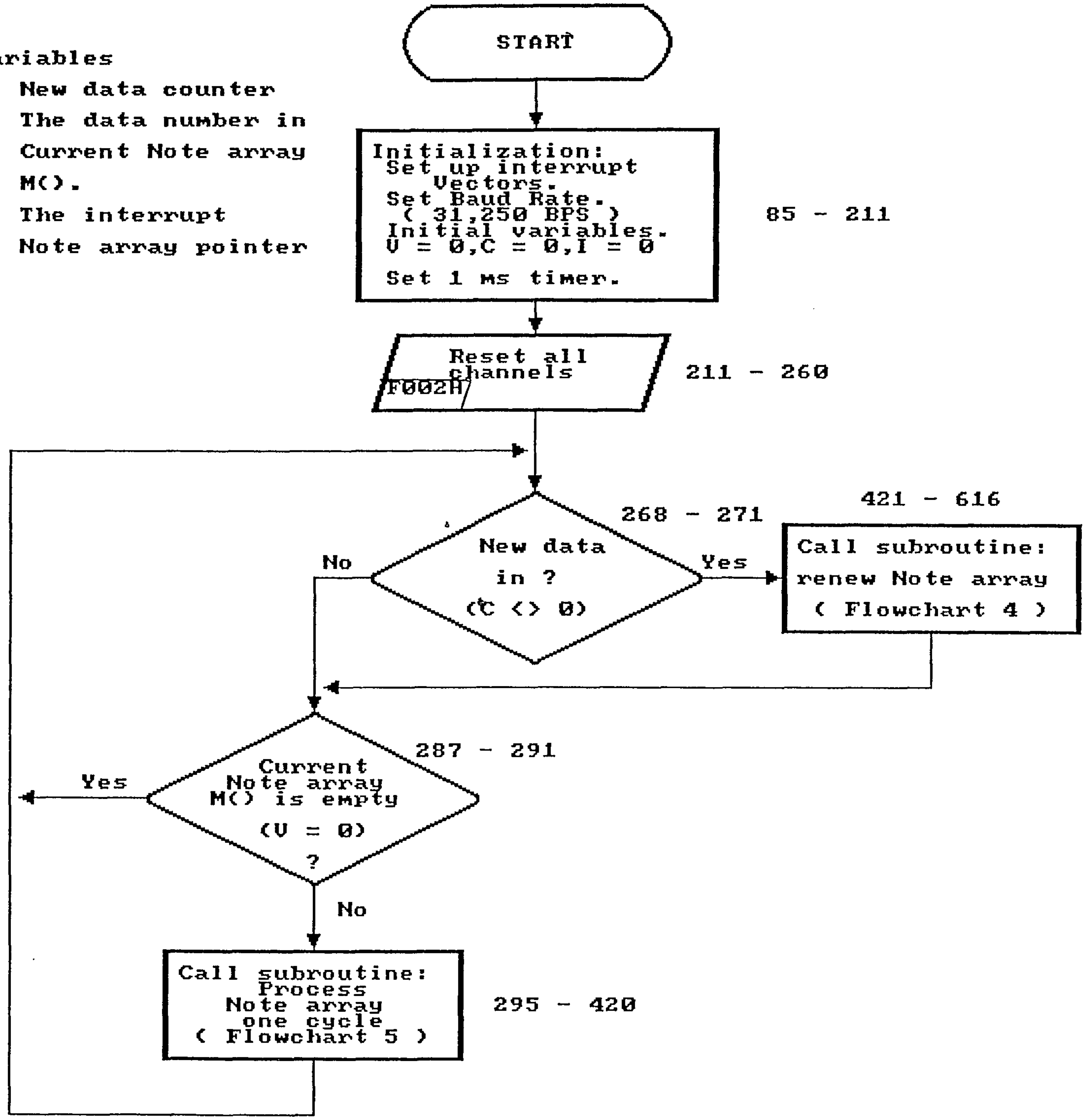
FLOWCHART 3 : MAIN PROGRAM

Variables

C: New data counter

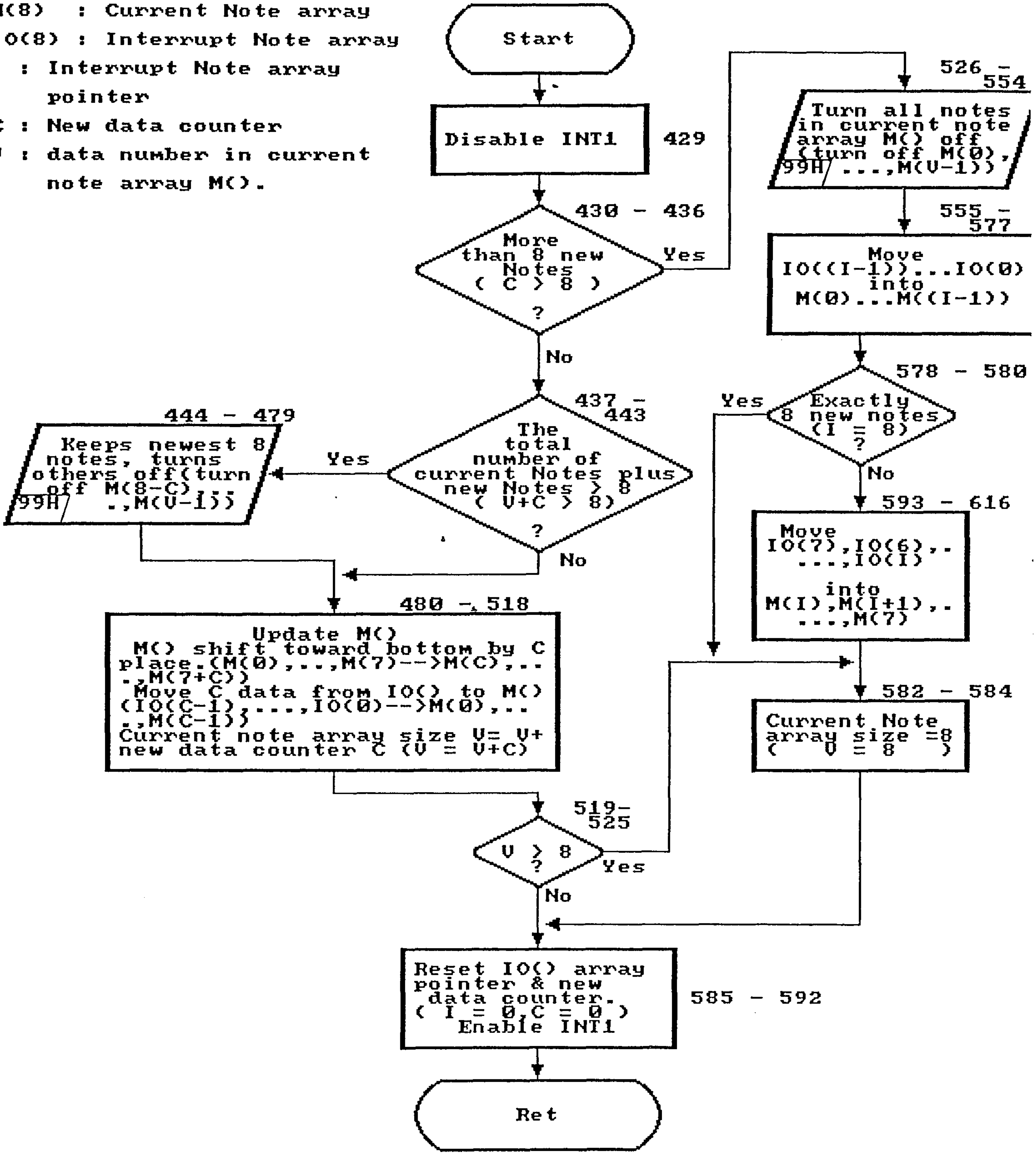
V: The data number in Current Note array M().

I: The interrupt Note array pointer



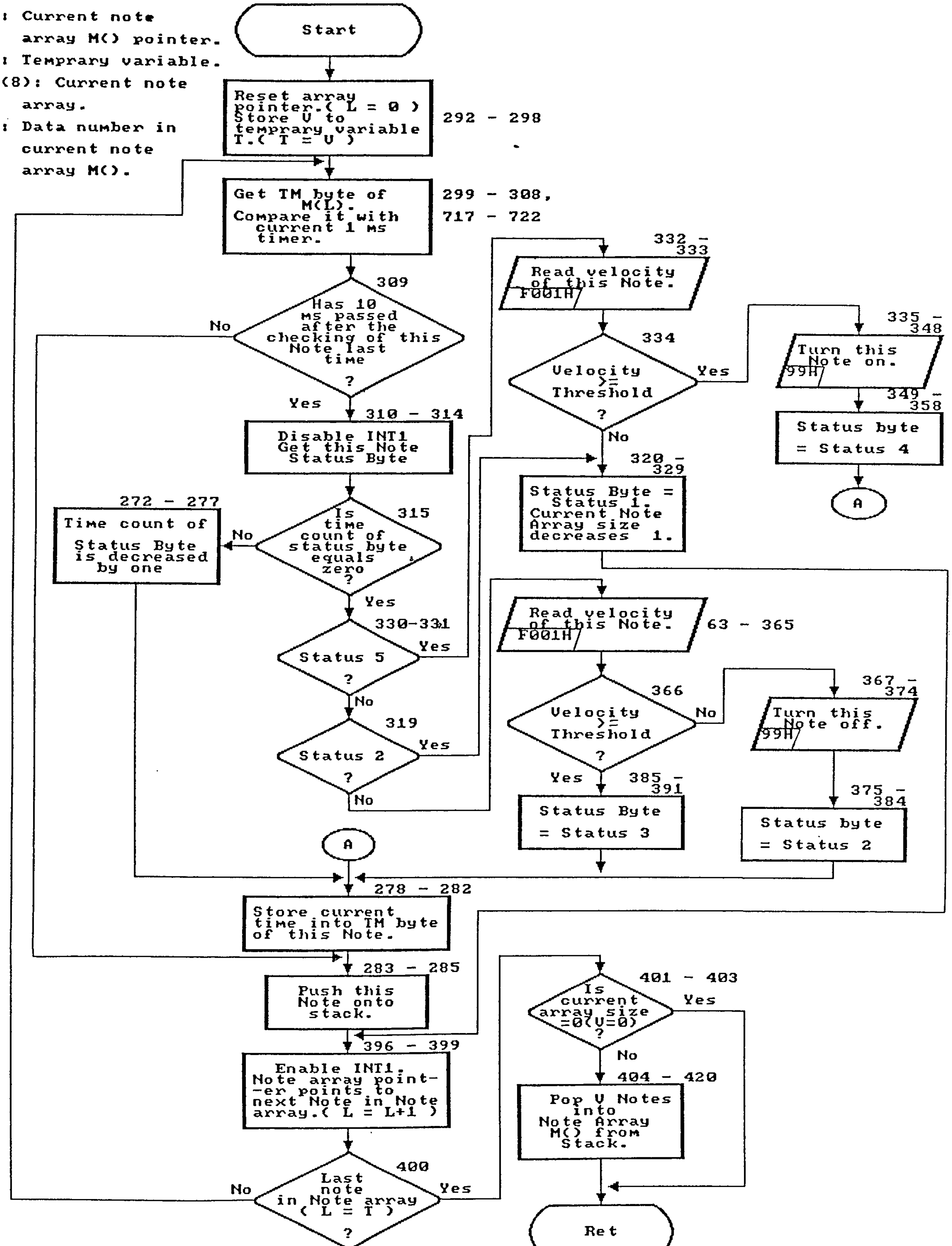
FLOWCHART 4 : RENEW NOTE ARRAY SUBROUTINE

M(8) : Current Note array
IO(8) : Interrupt Note array
I : Interrupt Note array pointer
C : New data counter
V : data number in current note array M().

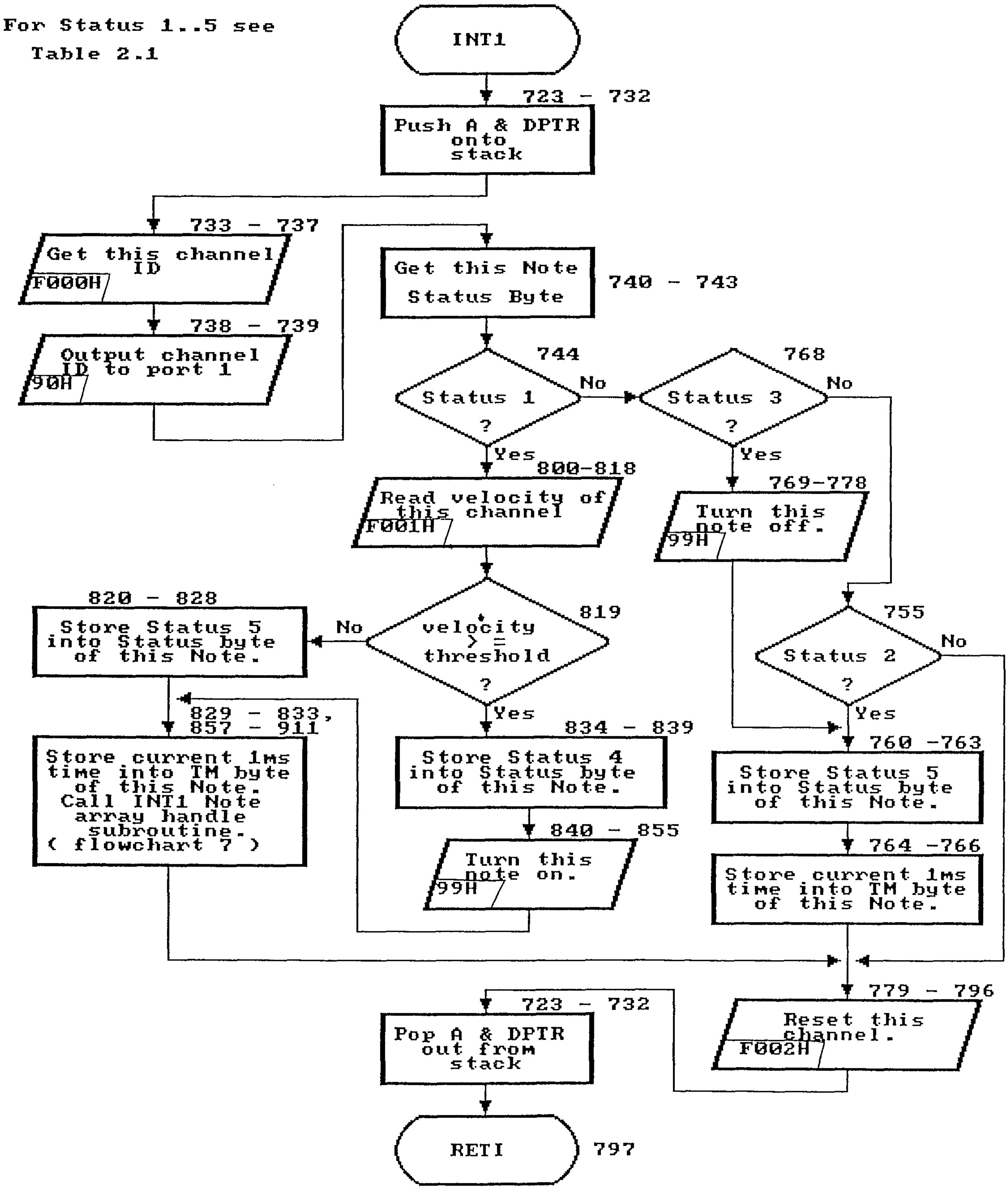


FLOWCHART 5 : PROCESSING NOTE ARRAY ONE CYCLE SUBROUTINE

L: Current note array M() pointer.
T: Temprary variable.
M(8): Current note array.
V: Data number in current note array M().

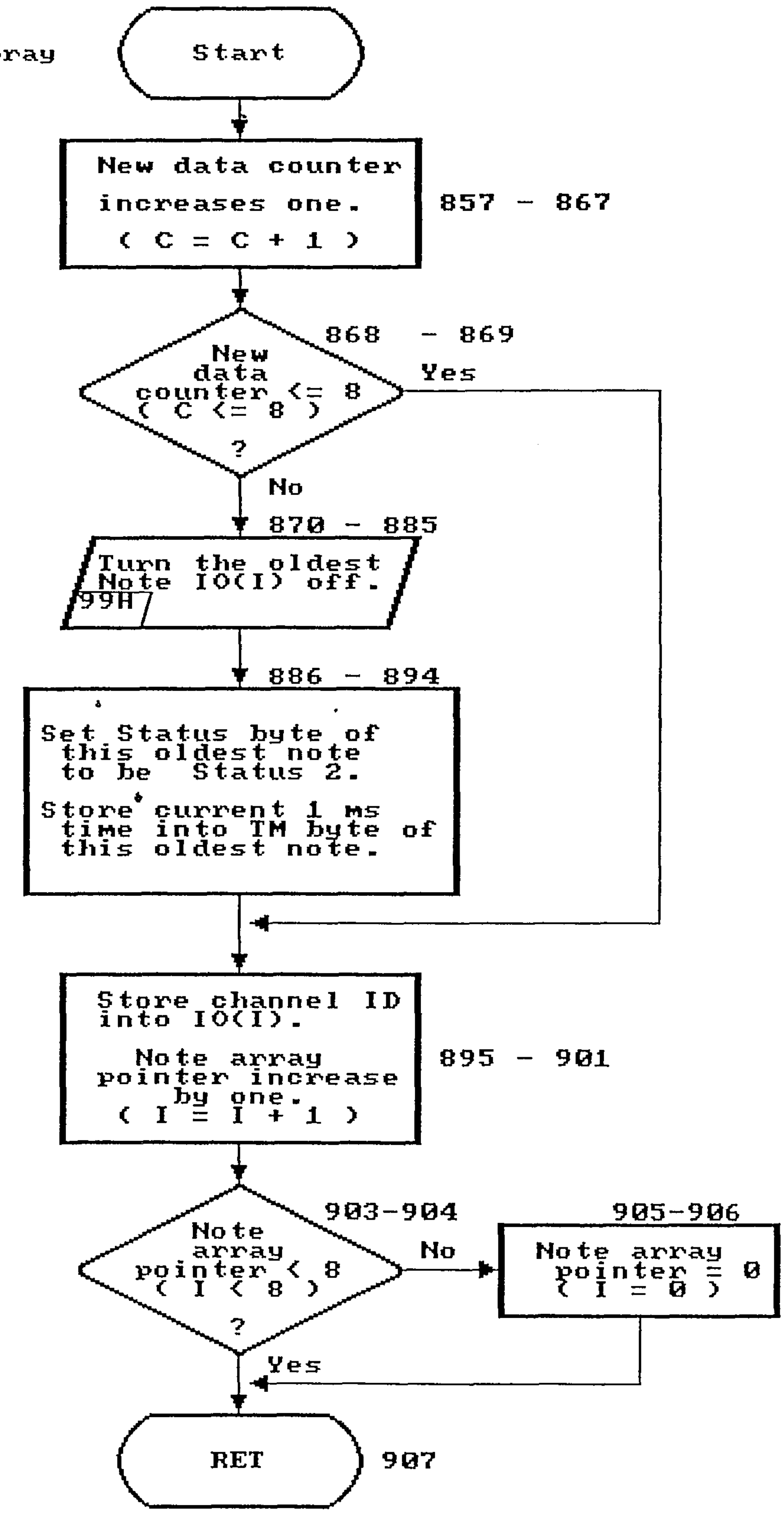


FLOWCHART 6 : INT1 INTERRUPT ROUTINE

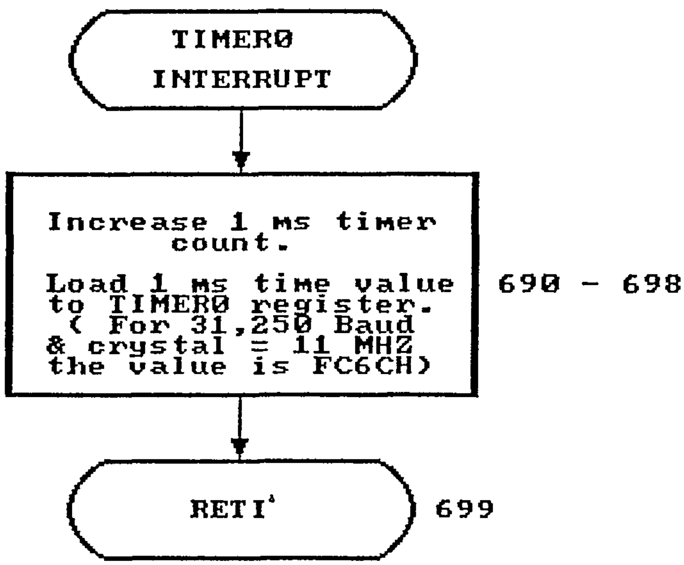


FLOWCHART 7 : INT1 NOTE ARRAY HANDLING SUBROUTINE

IO(8) : Interrupt Note array
I : Note array pointer
C : New data counter



FLOWCHART 8 : TIMER0 INTERRUPT ROUTINE
(1 MS TIMER FOR MIDIVIBS PROGRAM)



CHAPTER FOUR

TEST

A. TEST Program

The second program in Appendix B which is called MIDI DATA COLLECTION is a program that can record MIDI messages. It also can send out MIDI messages in the way just the same as the messages were recorded.

In the output file created by this program, there are two informations for each MIDI code. One is the MIDI code itself. The other is the time at which that code was recorded. This file starts at memory location 5000H.

Using this program we can know exactly MIDI code output sequence. By taking a look at MIDI file that was created by this program, we can know which MIDI code is sent, at what time.(The resolution is 1 ms.)

The flowcharts of this program, flowchart 9 and 10 are on page 42 and 43.

Before we use this test program to collect MIDI data, we need to clear MIDI data file area. The MIDI data file area is located from 5000H to 5FFFH. We can use Micromint command "FILL 5000H,5FFFH,0" to clear this area. After we have finished MIDI data collection, we push the push-button (S.W.1) in MIDI Data Collection Interface (Fig 3.2) to stop the collection program.

To send the MIDI messages which were recorded, it is simply by executing this MIDI Data Collection program. When the data area is not empty the program sends out the MIDI messages which were recorded automatically.

Each MIDI code in data file is represented by 4 bytes. The first byte is the MIDI code itself. The other 3 bytes are 1 ms time at which this MIDI code is recorded. When both MIDI code and 3 bytes of 1 ms time are zero this means the end of MIDI data file.

Following is the I/O configuration of the MIDI Data Collection Interface:

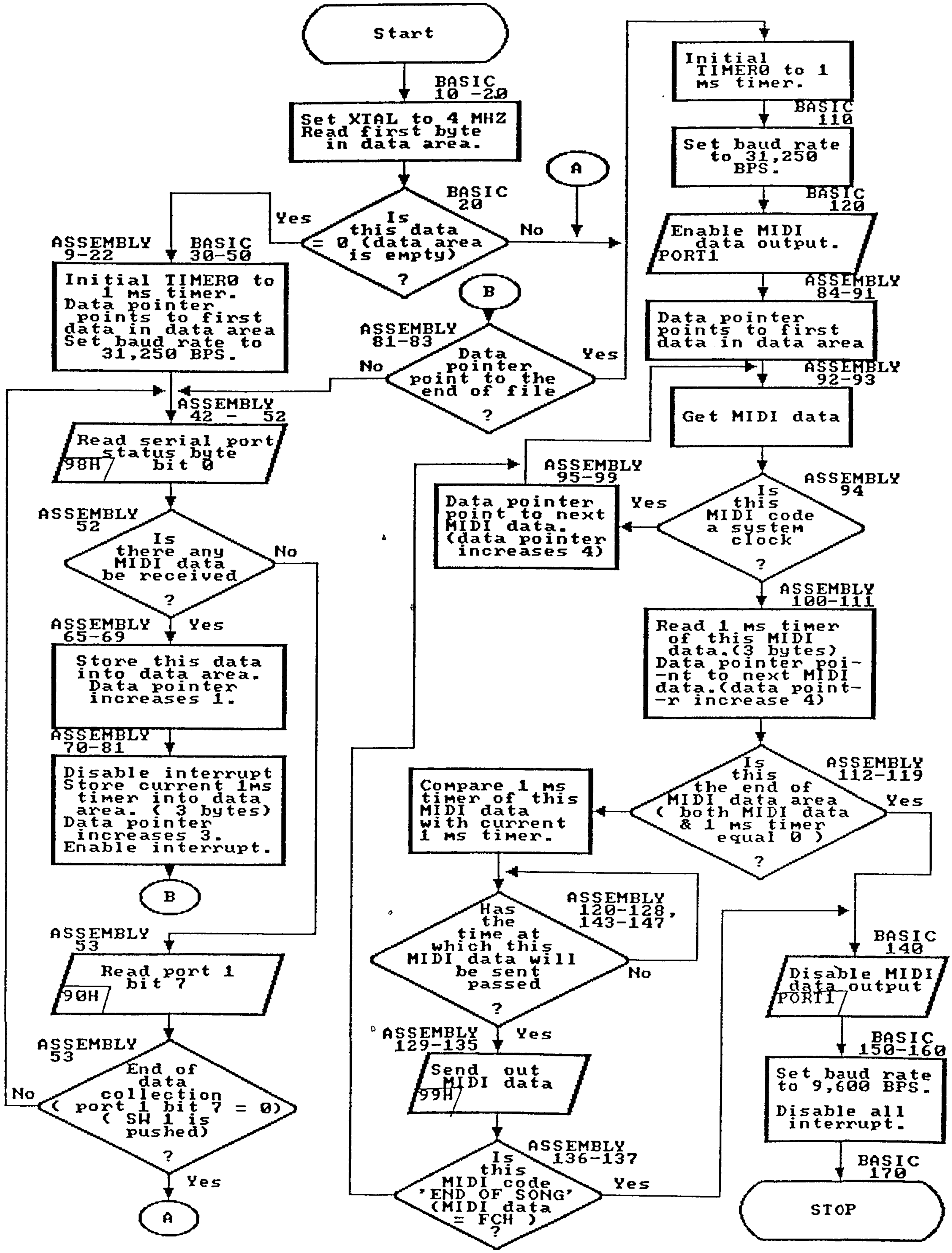
Port 1 bit 0 (Output) Disable/Enable the MIDI out. This bit is used as an output. When it is high the MIDI out transmission is disabled. When it is low the MIDI out transmission is enabled.

Port 1 bit 7 (Input) When SW 1 is pushed, this bit is low. When it is released, this bit is high.

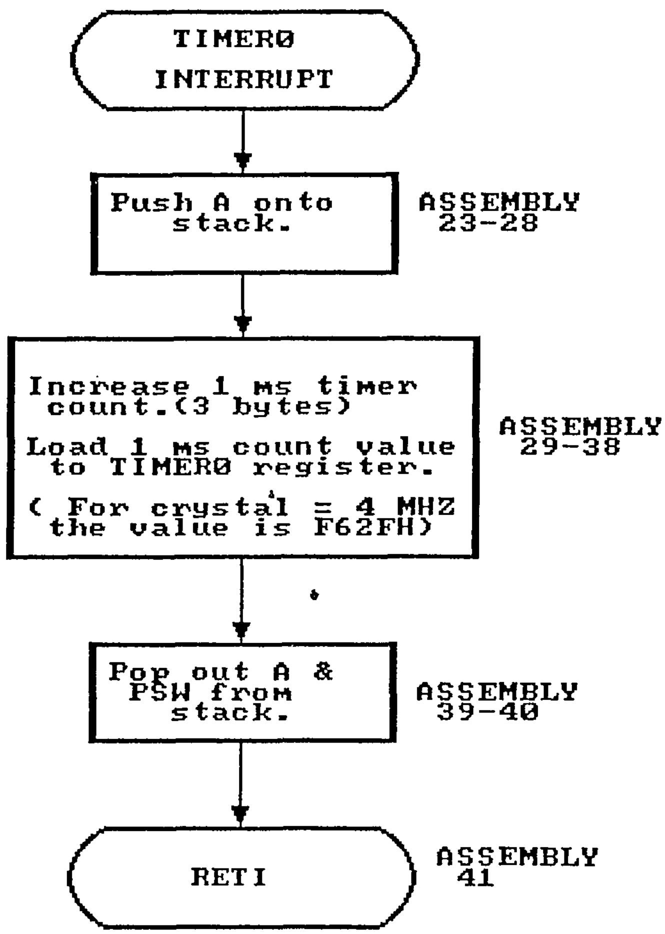
The MIDI IN is connected to serial in of SBC. The MIDI OUT is controlled by port 1 bit 0. When port 1 bit 0 is low the MIDI OUT is connected to serial out of SBC. When port 1 bit 0 is high the MIDI OUT is disabled.

B. FLOWCHARTS

FLOWCHART 9 : MIDI DATA COLLECTION PROGRAM



FLOWCHART 10 : TIMER0 INTERRUPT ROUTINE
(1 MS TIMER FOR MIDI DATA COLLECTION PROGRAM)



C. **Hardware Setup for Test Program**

The hardware is set up by connect vibraphone to the MIDIVIBS and then connect the MIDI output to the MIDI Data Collection Interface then the SBC board then the terminal. Fig 3.1 shows the system diagram of these connection. The circuit of MIDI Data Collection Interface is shown in Fig 3.2.

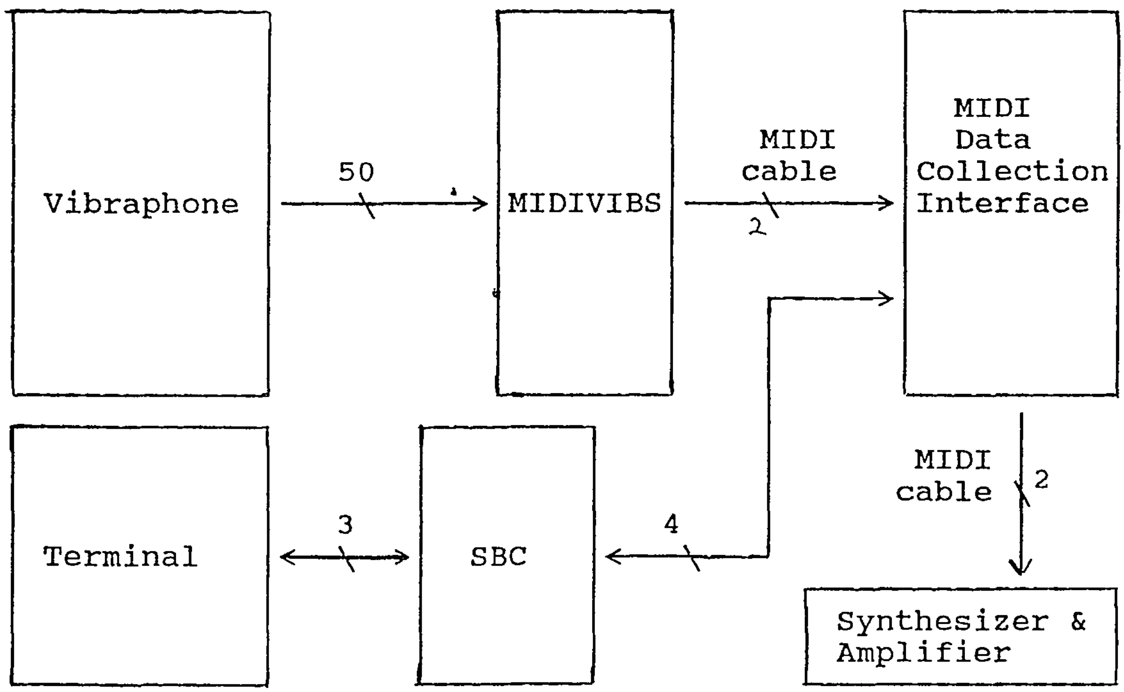


Fig 3.1 Diagram of test system setup

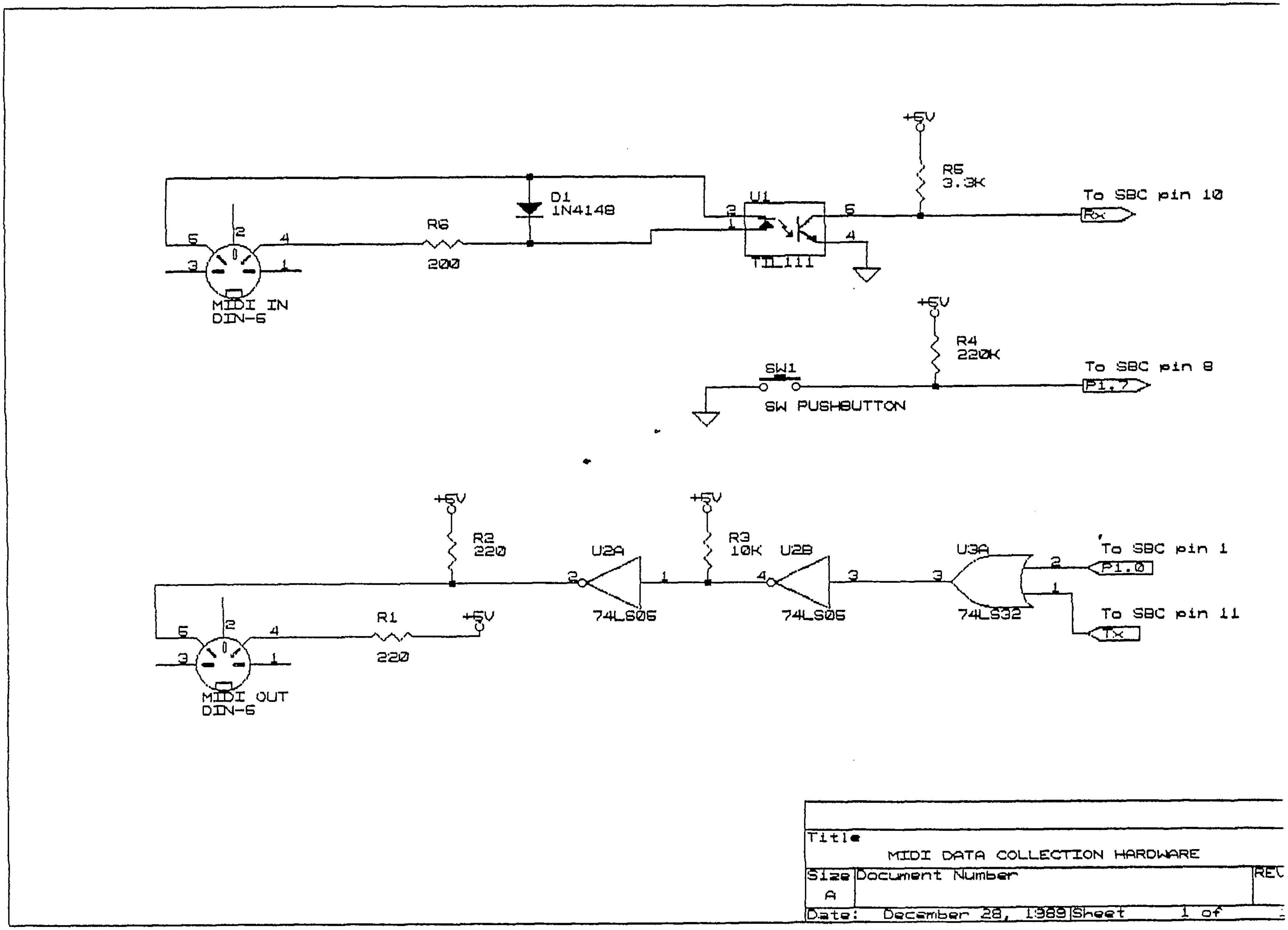


Fig 3.2 The circuit of MIDI Data Collection Interface

D. **Analysis of Test Result**

1. **Example 1**

The data of this example is collected by playing one note on the vibraphone. Fig 3.3 is the data of MIDI file produced by the MIDI DATA COLLECTION program and Fig 3.4 is the time diagram analysis of this MIDI file.

```
DIS 5000H,501FH
5000H   90H 00H 18H FAH 4BH 00H 18H FBH   . . . . K . . .
5008H   40H 00H 18H FBH 4BH 00H 22H 97H   @ . . . K . " .
5010H   00H 00H 22H 97H 00H 00H 00H 00H   . . " . . . . .
5018H   00H 00H 00H 00H 00H 00H 00H 00H   . . . . . . . .
READY
>
```

Fig 3.3 A simple example MIDI file

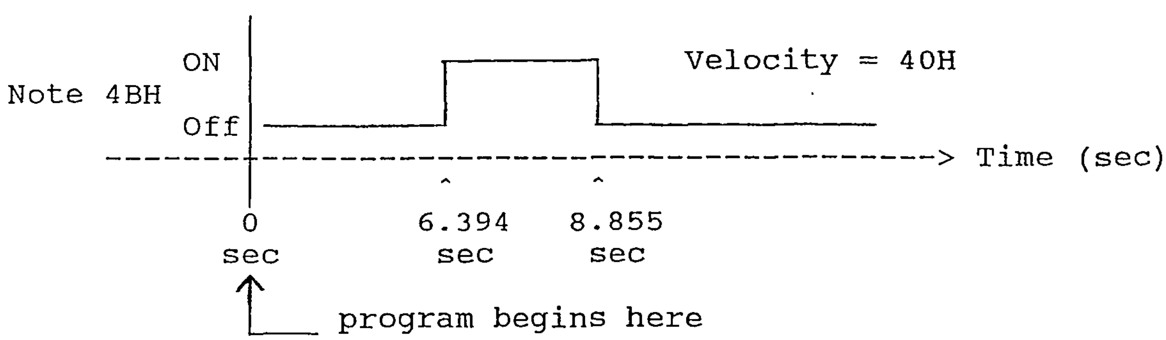


Fig 3.4 Time diagram of MIDI file in Fig 3.3

There are four bytes information in each MIDI data. The first byte is MIDI code itself as 90H in this example. The second, third and fourth bytes are the time at which this MIDI code is recorded. For the first MIDI code 90H, the time is 0018FAH ms which means the MIDI code 90H is recorded at time 6.394 sec counted from the time at which this program was executed.

For the second MIDI data the MIDI code is 4BH. The time is 0018FBH ms. Which means MIDI code 4BH is recorded at time 6.395 sec. Notice that the time resolution of this program is only 1 ms. The MIDI data is sent at 31,250 BPS. In 1 ms it is possible to have 3 bytes data been transmitted.

The first MIDI message is completed by the first 3 MIDI codes which are 90H, 4BH and 40H. This message means turn note 4BH on with velocity 40H. The codes of second MIDI message is 4BH and 00H Which means turn note 4BH off. Because the MIDIVIBS program uses the running status of MIDI protocol, so it doesn't send the status byte 90H when the MIDI message is turn Note off.

When both MIDI code and all 3 bytes of 1 ms time are zero that means the end of MIDI file.

2. Example 2

The data of this example is collected by trying to hit three Notes on the vibraphone almost at the same time. Fig 3.5 is the MIDI file produced by this MIDI DATA COLLECTION program and Fig 3.6 is the time diagram analysis of this MIDI file.

```
DIS 5000H,5080H

 5000H  90H 00H 2CH 61H 4EH 00H 2CH 61H   . . , a N . , a
 5008H  42H 00H 2CH 62H 90H 00H 2CH 6CH   B . , b . . , l
 5010H  52H 00H 2CH 6CH 47H 00H 2CH 6CH   R . , l G . , l
 5018H  90H 00H 2CH 6EH 50H 00H 2CH 6FH   . . , n P . , o
 5020H  40H 00H 2CH 6FH 4EH 00H 2DH 1FH   @ . , o N . - .
 5028H  00H 00H 2DH 1FH 50H 00H 2DH E5H   . . - . P . - .
 5030H  00H 00H 2DH E5H 52H 00H 2DH EDH   . . - . R . - .
 5038H  00H 00H 2DH EEH 00H 00H 00H 00H   . . - . . . . .
 5040H  00H 00H 00H 00H 00H 00H 00H 00H   . . . . . . . .

READY
>
```

Fig 3.5 MIDI file for example 2

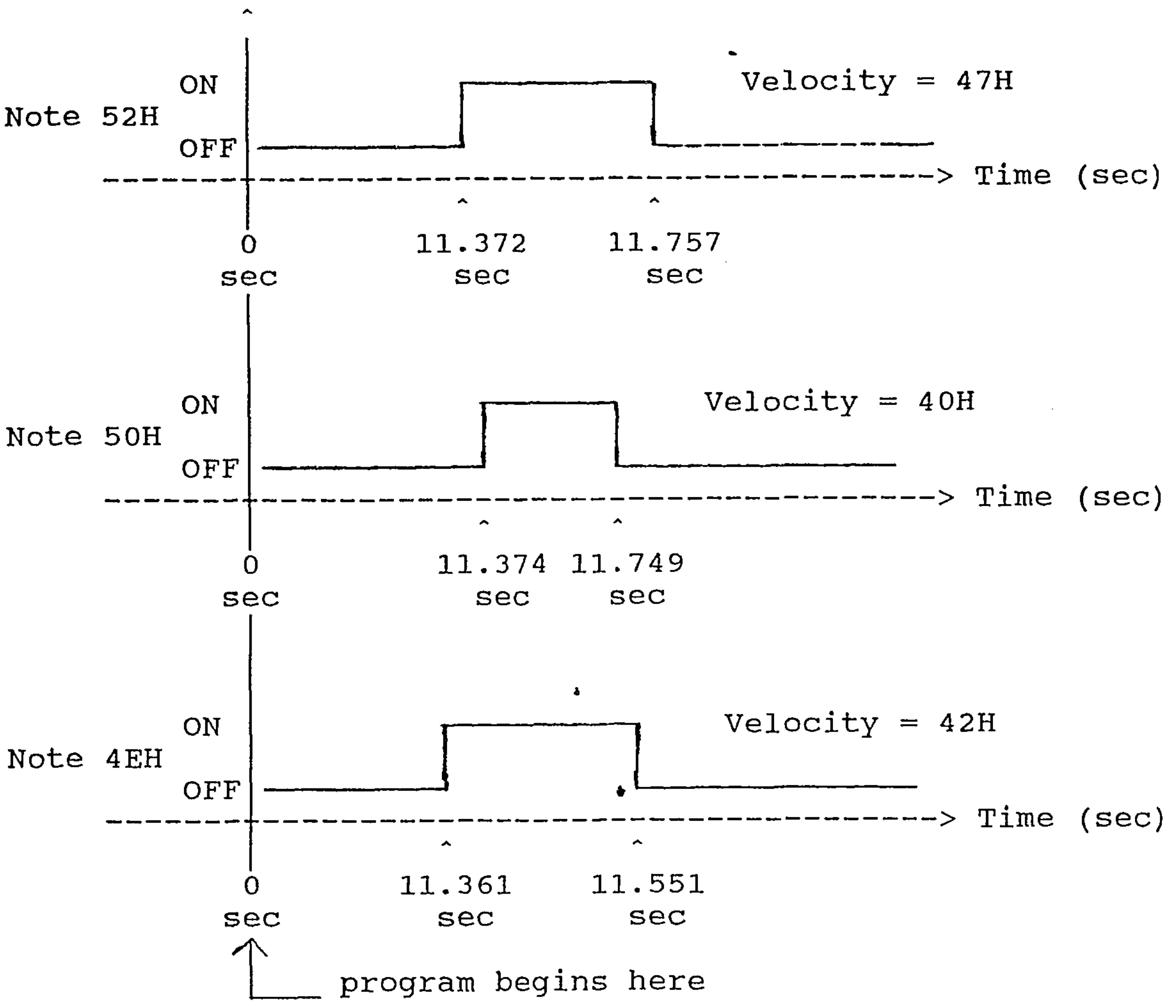


Fig 3.6 Time diagram of MIDI file in Fig 3.5

In this example there are three Notes 4EH, 50H and 52H be turned ON at time 11.361 sec, 11.374 sec and 11.372 sec respectively. They are turned OFF at time 11.551 sec, 11.749 and 11.757 respectively. This means that the interrupt system responses properly.

3. Example 3

In this example data is collected by playing a song on the vibraphone. Fig 3.7 is the MIDI file produced by this MIDI DATA COLLECTION program and Fig 3.8 is the time diagram analysis of this MIDI file.

```
DIS 5000H,5280H

5000H  90H 00H 08H EDH 4EH 00H 08H EEH  . . . . N . . .
5008H  2BH 00H 08H EEH 4EH 00H 09H E7H  + . . . N . . .
5010H  00H 00H 09H E7H 90H 00H 0AH 1AH  . . . . . . . .
5018H  4EH 00H 0AH 1AH 25H 00H 0AH 1AH  N . . . % . . .
5020H  4EH 00H 0AH EBH 00H 00H 0AH EBH  N . . . . . . .
5028H  90H 00H 0BH 14H 4EH 00H 0BH 15H  . . . . N . . .
5030H  28H 00H 0BH 15H 4EH 00H 0BH FAH  ( . . . N . . .
5038H  00H 00H 0BH FAH 90H 00H 0DH 55H  . . . . . . . U
5040H  4BH 00H 0DH 55H 30H 00H 0DH 55H  K . . U 0 . . U
5048H  4BH 00H 0EH 6EH 00H 00H 0EH 6EH  K . . n . . . n
5050H  90H 00H 0FH 83H 50H 00H 0FH 84H  . . . . P . . .
5058H  34H 00H 0FH 84H 50H 00H 10H D0H  4 . . . P . . .
5060H  00H 00H 10H D1H 90H 00H 10H FCH  . . . . . . . .
```

Fig 3.7 MIDI file for example 3 (Continued on next page)

5068H	50H	00H	10H	FCH	30H	00H	10H	FDH	P . . . 0 . . .
5070H	50H	00H	11H	B9H	00H	00H	11H	BAH	P . . . .
5078H	90H	00H	11H	E3H	50H	00H	11H	E3H	. . . . P . . .
5080H	33H	00H	11H	E4H	50H	00H	13H	26H	3 . . . P . . &
5088H	00H	00H	13H	26H	90H	00H	13H	F3H	. . . & . . . .
5090H	4EH	00H	13H	F3H	35H	00H	13H	F3H	N . . . 5 . . .
5098H	4EH	00H	15H	16H	00H	00H	15H	16H	N . . . .
50A0H	90H	00H	16H	28H	4EH	00H	16H	28H	. . . (N . . (
50A8H	31H	00H	16H	28H	4EH	00H	17H	4BH	1 . . (N . . K
50B0H	00H	00H	17H	4BH	90H	00H	17H	97H	. . . K . . . .
50B8H	50H	00H	17H	98H	31H	00H	17H	98H	P . . . 1 . . .
50C0H	90H	00H	18H	6FH	52H	00H	18H	6FH	. . . o R . . o
50C8H	39H	00H	18H	6FH	50H	00H	18H	DAH	9 . . o P . . .
50D0H	00H	00H	18H	DBH	52H	00H	19H	BCH	. . . . R . . .
50D8H	00H	00H	19H	BCH	90H	00H	1AH	9AH	. . . .
50E0H	52H	00H	1AH	9AH	3AH	00H	1AH	9BH	R . . . : . . .
50E8H	52H	00H	1BH	F1H	00H	00H	1BH	F2H	R . . . .
50F0H	90H	00H	1CH	CCH	51H	00H	1CH	CCH	. . . . Q . . .
50F8H	43H	00H	1CH	CCH	51H	00H	1EH	38H	C . . . Q . . 8
5100H	00H	00H	1EH	38H	90H	00H	1EH	3BH	. . . 8 . . . ;
5108H	50H	00H	1EH	3CH	38H	00H	1EH	3CH	P . . < 8 . . <
5110H	90H	00H	1FH	2BH	4FH	00H	1FH	2BH	. . . + O . . +

Fig 3.7 MIDI file for example 3 (Continued on next page)

5118H	3CH	00H	1FH	2CH	50H	00H	1FH	A8H	< . . , P . . .
5120H	00H	00H	1FH	A9H	4FH	00H	20H	8EH	. . . . O . .
5128H	00H	00H	20H	8EH	90H	00H	23H	AFH	. . . . . # .
5130H	4FH	00H	23H	B0H	3CH	00H	23H	B0H	O . # . < . # .
5138H	90H	00H	25H	06H	50H	00H	25H	06H	. . % . P . % .
5140H	34H	00H	25H	06H	4FH	00H	25H	07H	4 . % . O . % .
5148H	00H	00H	25H	08H	90H	00H	25H	F5H	. . % . . . % .
5150H	51H	00H	25H	F5H	38H	00H	25H	F6H	Q . % . 8 . % .
5158H	50H	00H	26H	68H	00H	00H	26H	68H	P . & h . . & h
5160H	51H	00H	27H	43H	00H	00H	27H	43H	Q . ' C . . ' C
5168H	90H	00H	28H	3DH	51H	00H	28H	3DH	. . (= Q . (=
5170H	39H	00H	28H	3EH	51H	00H	29H	8AH	9 . (> Q .) .
5178H	00H	00H	29H	8AH	90H	00H	2AH	7EH	. .) . . . * ~
5180H	50H	00H	2AH	7FH	37H	00H	2AH	7FH	P . * 7 . *
5188H	50H	00H	2BH	D6H	00H	00H	2BH	D6H	P . + . . . + .
5190H	90H	00H	2CH	05H	4FH	00H	2CH	06H	. . , . O . , .
5198H	31H	00H	2CH	06H	90H	00H	2CH	F9H	1 . , . . . , .
51A0H	50H	00H	2CH	F9H	3AH	00H	2CH	FAH	P . , . : . , .
51A8H	4FH	00H	2DH	3EH	00H	00H	2DH	3EH	O . - > . . - >
51B0H	50H	00H	2EH	65H	00H	00H	2EH	65H	P . . e . . . e

Fig 3.7 MIDI file for example 3 (Continued on next page)

```
 51B8H   90H 00H 2FH 42H 4EH 00H 2FH 42H   . . / B N . / B
 51C0H   30H 00H 2FH 42H 4EH 00H 30H 50H   0 . / B N . 0 P
 51C8H   00H 00H 30H 51H 90H 00H 31H 9AH   . . 0 Q . . 1 .
 51D0H   4EH 00H 31H 9BH 35H 00H 31H 9BH   N . 1 . 5 . 1 .
 51D8H   4EH 00H 32H BEH 00H 00H 32H BEH   N . 2 . . . 2 .
 51E0H   90H 00H 32H E2H 50H 00H 32H E2H   . . 2 . P . 2 .
 51E8H   36H 00H 32H E3H 90H 00H 33H E8H   6 . 2 . . . 3 .
 51F0H   4FH 00H 33H E8H 3BH 00H 33H E9H   O . 3 . ; . 3 .
 51F8H   50H 00H 34H 4EH 00H 00H 34H 4FH   P . 4 N . . 4 O
 5200H   4FH 00H 35H 3FH 00H 00H 35H 40H   O . 5 ? . . 5 @
 5208H   90H 00H 36H 14H 4BH 00H 36H 14H   . . 6 . K . 6 .
 5210H   31H 00H 36H 14H 4BH 00H 37H 37H   1 . 6 . K . 7 7
 5218H   00H 00H 37H 37H 90H 00H 38H 6DH   . . 7 7 . . 8 m
 5220H   4DH 00H 38H 6DH 37H 00H 38H 6DH   M . 8 m 7 . 8 m
 5228H   90H 00H 39H CBH 4FH 00H 39H CCH   . . 9 . O . 9 .
 5230H   3DH 00H 39H CCH 4DH 00H 39H CFH   = . 9 . M . 9 .
 5238H   00H 00H 39H CFH 4FH 00H 3BH 23H   . . 9 . O . ; #
 5240H   00H 00H 3BH 23H 90H 00H 3BH 29H   . . ; # . . ; )
 5248H   4EH 00H 3BH 29H 38H 00H 3BH 29H   N . ; ) 8 . ; )
 5250H   4EH 00H 3CH 80H 00H 00H 3CH 80H   N . < . . . < .
 5258H   00H 00H 00H 00H 00H 00H 00H 00H   . . . . . . . .
READY
>
```

Fig 3.7 MIDI file for example 3

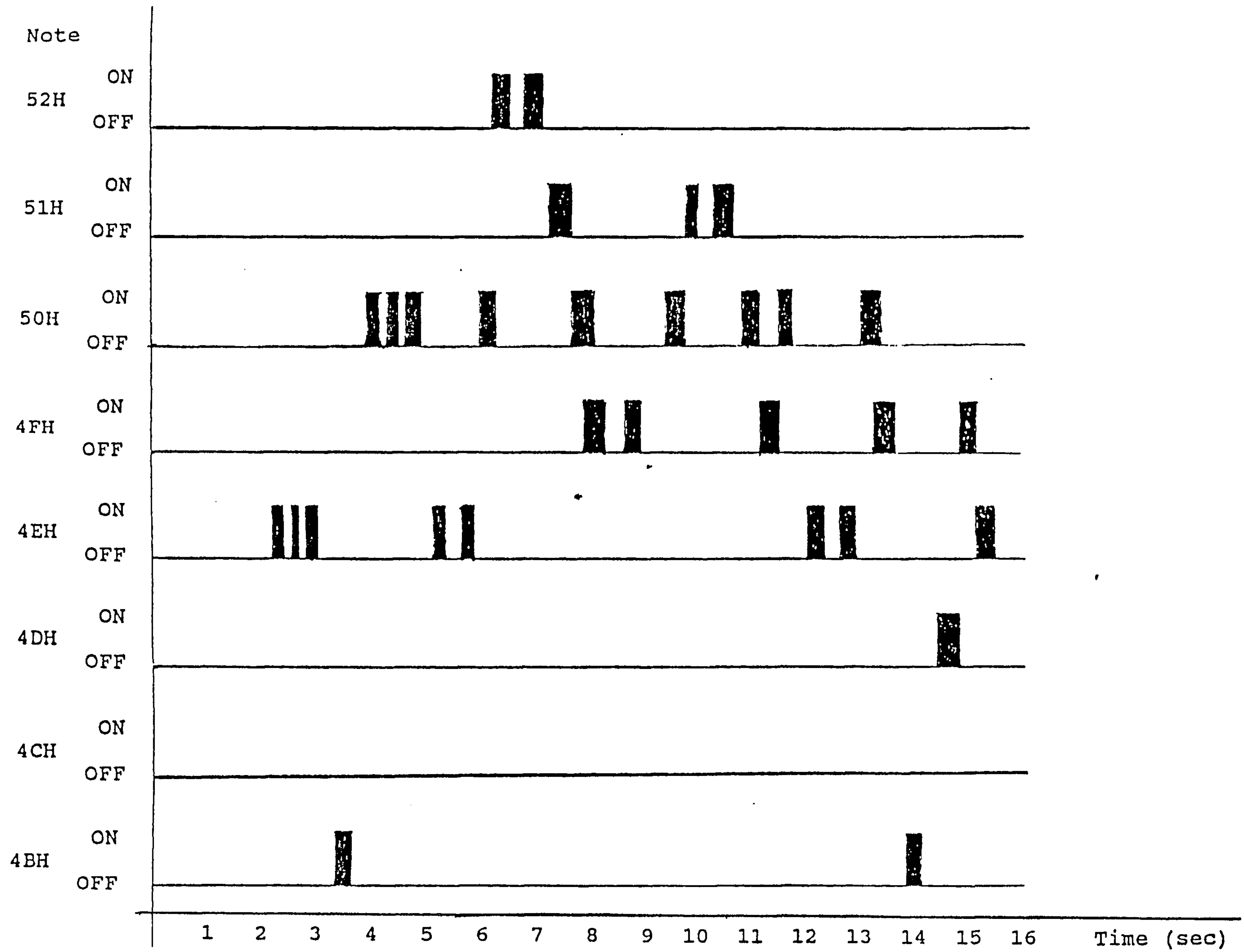


Fig 3.8 Time diagram of MIDI file in Fig 3.7

APPENDIX A

INTRODUCTION TO MIDI

INTRODUCTION TO MIDI

The MIDI is an abbreviation of Musical Instrument Digital Interface. By using MIDI protocol musical performance and other information can be transmitted and received by instruments using the common serial interface. The Baud Rate of this serial interface is 31.25K BAUD.

MIDI messages which transmit information between MIDI devices determine what kinds of musical events can be passed from device to device. It usually takes two or three bytes to send a MIDI message.

The first byte of any MIDI message is called the status byte. It tells what kind of message it is. The status byte might identify the message as a Note On message (one that tells about a note that just started),Note Off message (one that tells the end of a note), or any number of other possible types.

The bytes that follow the status byte are called data bytes. Each data byte elaborates on the information given by the status byte. For example, the first data byte in a Note On message tells the pitch of the note, and the second data byte tells the attack velocity of the note so that a MIDI device can tell how loud to play it.

To distinguish data byte from status byte, MIDI uses bytes that MSB is 0 as data bytes and bytes that MSB is 1 as status bytes. Many types of MIDI messages use two data bytes to carry additional information; some need only one data byte; still others use no data bytes at all.

There are many different kinds of MIDI messages as shown in FIG 4.1 and Fig 4.2.

One important feature about MIDI is that MIDI devices can use a technique called running status to make data transmission even faster. Running status allows a device to send a stream of messages of the same kind without repeating the status byte for each message. When the sending device wants to send another kind of message, it simply stops sending data and send another message as it would normally. By eliminating repeated status bytes, running status reduces the number of message bytes and speeds up note transmission.

The MIDI message used by this thesis is only the Note On command. The Note OFF command is replaced by a Note ON command with zero attack velocity. Also the program use the running status technique to speed up the processing.

- MIDI Messages
  - Channel
    - Voice
      - Note Off
      - Note On
      - Ployphonic key Pressure
      - Control Change (see Fig 4.2)
      - Program Change
      - Channel pressure
      - Pitch Bend Change
    - Mode
      - Reset All Controllers
      - Local Control
      - All Notes Off
      - Omni Mode Off
      - Omni Mode On
      - Mono Mode On
      - Poly Mode On
  - System
    - System exclusive messages
    - Common
      - MIDI Time Code Quarter Frame
      - Song Position Pointer
      - Song Select
      - Tune Request
      - EOX
    - Real-time
      - Timing Clock
      - Start
      - Continue
      - Stop
      - Active Sensing
      - System Reset

Fig 4.1 The different MIDI messages, arranged by message type.

Control Change

- Modulation Wheel or Level
- Breath Controller
- Foot Controller
- Portamento Time
- Data entry MSB
- Main volume
- Balance
- Pan
- Expression Controller
- General Purpose Controllers
- LSB for above Controllers in this figure
- Damper Pedal (sustain)
- Portamento
- Sostenuto
- Soft Pedal
- Hold 2
- External Effects Depth
- Tremolo Depth
- Chorus Depth
- Celeste (Detune) Depth
- Phaser Depth
- Data increment
- Data decrement
- Non-Registered Parameter Number
  - LSB
  - MSB
- Registered Parameter Number LSB
  - Pitch Bend Sensitivity
  - Fine Tuning
  - Coarse Tuning
- Registered Parameter Number MSB
  - Pitch Bend Sensitivity
  - Fine Tuning
  - Coarse Tuning

Fig 4.2 The different controllers under MIDI Control Change message.

APPENDIX B

Programs List

Program Page

MIDIVIBS Program. 61

MIDI Data Collection Program. 84

```
;CODE AREA BEGINS AT 2000H ( LENGTH 700H )
;  PROGRAM BEGINS AT 2090H
;
CODE_AREA EQU 2000H ;START ADDRESS OF CODE AREA
;
;
;-------------------------------------------------------------------;
; DATA_AREA MEMORY MAP  ( At least 1K memory begins at 4000H )      ;
;     C : 4100H   NEW DATA COUNTER                                  ;
;     I : 4101H   INTERRUPT NOTE ARRAY POINTER                      ;
;     T : 4102H - TEMPORARY ARRAY SIZE                              ;
;     V : 4103H - CURRENT ARRAY SIZE (DATA NUMBER IN M(8) ARRAY )   ;
; M(8): 4110H - 4117H  CURRENT NOTE ARRAY                           ;
; IO(8): 4120H - 4127H  INTERRUPT NOTE ARRAY                        ;
;                                                                   ;
;         41XXH   GSPE-NOTE TRANSFER TABLE                          ;
;         42XXH   Notes TM area                                     ;
;         43XXH   Notes status area                                 ;
;     ( XX IS A NUMBER FROM 32H TO 77H )                            ;
;                                                                   ;
;-------------------------------------------------------------------;
;
;
THREVEL1 EQU 04H    ;Noise recover threshold
THREVEL2 EQU 01H    ;Turn note off threshold
;
BAUD_COUNTER EQU 0FFF5H ;COUNTER FOR 31,250 BPS (AT 11 MHZ CRYSTAL)
TIM0_C_L EQU 6CH    ;1 MS TIMER COUNT LOW BYTE   (AT 11 MHZ CRYSTAL)
TIM0_C_H EQU 0FCH   ;1 MS TIMER COUNT HIGH BYTE  (AT 11 MHZ CRYSTAL)
;
S_2 EQU 13H         ;TURN OFF AT LEAST 30 MS
S_4 EQU 35H         ;TURN ON AT LEAST 50 MS
S_5 EQU 43H         ;TURN ON 30 MS LATER
;
;-------------------------------------------------------------------;
;                 GSPE --> KEY NO. TRANSFER TABLE                   ;
;      THIS TABLE IS SET UP BY INITIALIZATION SOFTWARE              ;
;-------------------------------------------------------------------;
;             ORG 4132H                   ;KEY CARD #5
;                     DB 35H          ;F 4
;                     DB 36H          ;F#4
;                     DB 37H          ;G 4
;                     DB 38H          ;G#4
;                     DB 39H          ;A 4
;                     DB 3AH          ;A#4
```

```
;               ORG 4140H                       ;KEY CARD #4
;                       DB 3BH               ;B 4
;                       DB 3CH               ;C 5
;                       DB 3DH               ;C#5
;                       DB 3EH               ;D 5
;                       DB 3FH               ;D#5
;                       DB 40H               ;E 5
;                       DB 41H               ;F 5
;                       DB 42H               ;F#5
;               ORG 4150H                       ;KEY CARD #3
;                       DB 43H               ;G 5
;                       DB 44H               ;G#5
;                       DB 45H               ;A 5
;                       DB 46H               ;A#5
;                       DB 47H               ;B 5
;                       DB 48H               ;C 6
;                       DB 49H               ;C#6
;                       DB 4AH               ;D 6
;               ORG 4160H                       ;KEY CARD #2
;                       DB 4BH               ;D#6
;                       DB 4CH               ;E 6
;                       DB 4DH               ;F 6
;                       DB 4EH               ;F#6
;                       DB 4FH               ;G 6
;                       DB 50H               ;G#6
;                       DB 51H               ;A 6
;                       DB 52H               ;A#6
;               ORG 4170H                       ;KEY CARD #1
;                       DB 53H               ;B 6
;                       DB 54H               ;C 7
;                       DB 55H               ;C#7
;                       DB 56H               ;D 7
;                       DB 57H               ;D#7
;                       DB 58H               ;E 7
;                       DB 59H               ;F 7
;
;-----------------------------------------------------------------;
;           INITIALIZATION BEFORE MAIN PROGRAM                    ;
;-----------------------------------------------------------------;
ORG CODE_AREA
        DB 0H
        DB 0AAH         ;TELL BASIC THAT RESET IS EXTERNAL
INITIAL EQU CODE_AREA+90H
ORG INITIAL
```

```
        ;-----------------------------------------------;
        ;           COLD RESET                          ;
        ;-----------------------------------------------;
                                ;First clear the internal memory
        MOV R0,#0FFH            ;Load R0 with the top of internal memory
        CLR A                   ;Set accumulator = 0
        ;
RESET1:
        MOV @R0,A               ;Loop until all the internal RAM
        DJNZ R0,reset1          ; RAM is cleared
        ;
                                ;Now set up the stack pointer and the
                                ; stack pointer holding register
        MOV     SP,#4DH        ;4DH is the initialized value of the stack
        MOV     3EH,#4DH        ;This is the SP holding register
        ;
        MOV DPTR,#4000H          ;Clear DATA_AREA
LAB203: MOV A,#0
        MOVX @DPTR,A
        INC DPTR
        MOV A,83H
        CJNE A,#44H,LAB203

        ;-----------------------------------------------;
        ;           SET UP INTERRUPT VECTOR             ;
        ;-----------------------------------------------;

                                ;Now set up interrupt vectors
        MOV DPTR,#400BH
        MOV A,#02H              ;LJMP machine code
        MOVX @DPTR,A
        INC DPTR
        MOV A,#HIGH TMINT       ;Time 0 interrupt routine high byte
        MOVX @DPTR,A            ;  address
        INC DPTR
        MOV A,#LOW TMINT        ;Time 0 interrupt routine low byte
        MOVX @DPTR,A            ;  address
        ;
        MOV DPTR,#4013H
        MOV A,#02H              ;LJMP machine code
        MOVX @DPTR,A
        INC DPTR
        MOV A,#HIGH INT_1       ;INT 1 interrupt routine high byte
        MOVX @DPTR,A            ;  address
        INC DPTR
```

```
        MOV A,#LOW INT_1    ;INT 1 interrupt routine low byte
        MOVX @DPTR,A        ;  address
        ;
        ;-------------------------------------------------;
        ;           CHECK IF RAM OK                       ;
        ;-------------------------------------------------;

        MOV DPTR,#400BH
        MOVX A,@DPTR
        CJNE A,#02H,RAM_ERROR         ;Check LJMP machine code
        INC DPTR
        MOVX A,@DPTR
        CJNE A,#HIGH TMINT,RAM_ERROR  ;Check TIMER interrupt vector
        INC DPTR
        MOVX A,@DPTR
        CJNE A,#LOW TMINT,RAM_ERROR   ;Check TIMER interrupt vector
        ;
        MOV DPTR,#4013H
        MOVX A,@DPTR
        CJNE A,#02H,RAM_ERROR         ;Check LJMP machine code
        INC DPTR
        MOVX A,@DPTR
        CJNE A,#HIGH INT_1,RAM_ERROR  ;Check INT interrupt vector
        INC DPTR
        MOVX A,@DPTR
        CJNE A,#LOW INT_1,RAM_ERROR   ;Check INT interrupt vector
        SJMP SETTABLE
RAM_ERROR:                            ;If RAM error loop here
        SJMP RAM_ERROR                ; forever

        ;-------------------------------------------------;
        ;          SET UP GSPE-NOTE TRANSFER TABLE        ;
        ;-------------------------------------------------;
        ;
SETTABLE:
        MOV DPTR,#4132H       ;Start from #4132H
        MOV 18H,#34H          ;FIRST NOTE
LAB410:
        INC 18H
        MOV A,18H
        MOVX @DPTR,A
        INC 82H               ;Point to next channel
        MOV A,82H
        ANL A,#0FH
        XRL A,#08H
```

```
        JNZ LAB410
        MOV A,82H
        ADD A,#8H           ;Point to next key card
        MOV 82H,A
        CJNE A,#80H,LAB410
        ;
        ;--------------------------------------------------;
        ;           SET BAUD RATE                          ;
        ;--------------------------------------------------;
        ;
        MOV RCAP2H,#HIGH BAUD_COUNTER
        MOV RCAP2L,#LOW  BAUD_COUNTER
        ;
        ;--------------------------------------------------;
        ;            SET INT1 TO LEVEL TRIGGER             ;
        ;--------------------------------------------------;

        MOV 88H,#050H       ;REM SET INT1 TO LEVEL TRIGGER
        ;
        ;--------------------------------------------------;
        ;           SET TIMER0 TO 1 MS TIMER               ;
        ;--------------------------------------------------;

        MOV 18H,#0H         ;Reset timer (1 ms/count)
        ANL 88H,#0CFH       ;Disable timer0
        MOV 8AH,#06CH   ;Set 1 ms timer0 low byte FOR XTAL = 11 MHZ
        MOV 8CH,#0FCH       ;Set 1 ms timer0 high byte
        ANL 89H,#0F0H       ;Set timer0 to 16 bit timer
        ORL 89H,#01H        ;   i.e. MODE 1
        ORL 88H,#30H        ;Enable timer0
        ORL 0A8H,#82H       ;Enable timer0 interrupt
       ;--------------------------------------------------;
       ;          RESET ALL CHANNELS                      ;
       ;--------------------------------------------------;
        MOV DPTR,#0F002H      ;Reset address 0F002H
        MOV 1BH,#38H          ;Start from #38H
LAB202: MOV 90H,1BH           ;Out to port 1
        MOV 1CH,#0FEH         ;Loop 2 counter  (count 2 times)
LAB201: MOV A,#0              ;Loop 1 counter  (count 256 times)
LAB204: MOVX @DPTR,A          ;Reset
        INC A
        JNZ LAB204            ;Loop 1
        INC 1CH
        MOV A,1CH
        JNZ LAB201            ;Loop 2
```

```
        INC 1BH                 ;Point to next channel
        MOV A,1BH
        ANL A,#0FH
        JNZ LAB202
        MOV A,1BH
        ADD A,#8H             ;Point to next key card
        MOV 1BH,A
        CJNE A,#88H,LAB202
       ;------------------------------------------------;
       ;          MAKE SURE ALL CHANNELS                ;
       ;                ARE RESET                       ;
       ;------------------------------------------------;
LAB208: MOV DPTR,#0F000H
        MOVX A,@DPTR
        MOV 1BH,A             ;Read GSPE
        ANL A,#0F0H
        JZ LAB207             ;If GSPE = 0F0H that means all channels
        MOV A,1BH             ; have been reset JMP to LAB207
        ADD A,#8H
        MOV 90H,A             ;Output Channel to port 1
        MOV DPTR,#0F002H      ;Send out reset signal
        MOV A,#0
LAB209: MOVX @DPTR,A  ;Reset this particular channel 256 times
        INC A
        JNZ LAB209
        SJMP LAB208
;
LAB207:
;
        MOV A,#0
        MOV DPTR,#0F001H
        MOVX @DPTR,A          ;Reset A/D converter
        MOV DPTR,#0F001H      ;Read A/D converter to turn
        MOVX A,@DPTR          ; the overflow LED off
;
;------------------------------------------------------------------;
;                          MAIN PROGRAM                            ;
;------------------------------------------------------------------;
;
MAIN_PROGRAM:

        ORL 0A8H,#84H              ;Enable INT1 interrupt
LAB301: MOV DPTR,#4100H
        MOVX A,@DPTR       ;Check C -> new data counter
        JZ LAB300          ;If C = 0 jmp to LAB300
```

```
             LJMP LAB302        ;C <> 0 jmp to LAB302
     LABS20:
             MOV 83H,#43H
             MOV 82H,1DH
             MOVX A,@DPTR           ;Get this Note status byte
             DEC A                  ;Status time count decreases 1
             MOVX @DPTR,A
             MOV 83H,#42H
             MOV 82H,1DH            ;Store current time -> 42xxH
             MOV A,18H              ; where xx is channel ID (GSPE)
             MOVX @DPTR,A
     LAB306: PUSH 1DH           ;Push GSPE
             LJMP LAB309
     ;
     LAB300: MOV 82H,#03H   ;
             MOVX A,@DPTR   ;Check V if V <> 0 then jmp to LAB305
             JNZ LAB305     ;
             LCALL C_STATUS
             SJMP LAB301
     ;
     ;   V <> 0
     ;
     LAB305:
             DEC 82H
             MOVX @DPTR,A             ;STORE V TO T
             MOV 20H,A                ;INITIAL LOOP COUNTER L
     LAB307: MOV A,20H
             ADD A,#0FH
             MOV 83H,#41H
             MOV 82H,A
             MOVX A,@DPTR           ;Get M(L)
             MOV 1DH,A              ;GSPE (key no.)
             MOV 83H,#42H
             MOV 82H,1DH
             MOVX A,@DPTR           ;Get last check time
             LCALL CHKTM            ;Check if 10 ms passed
             JC LAB306              ;If not yet jmp to LAB306
             ANL 0A8H,#0FBH         ;Disable INT1
             MOV 83H,#43H
             MOV 82H,1DH
             MOVX A,@DPTR          ;Get this Note status byte
             ANL A,#0FH            ;GET STATUS TIME COUNT NIBBLE
             JNZ LABS20            ;Not yet -> status time count - 1
```

```
        MOV 83H,#43H
        MOV 82H,1DH
        MOVX A,@DPTR
        CJNE A,#10H,LABS15
LABS155:
        MOV 83H,#43H
        MOV 82H,1DH
        MOV A,#0
        MOVX @DPTR,A              ;Status  S 2 -> S 1
        MOV DPTR,#4103H           ;
        MOVX A,@DPTR              ; V = V - 1
        DEC A                     ;
        MOVX @DPTR,A              ;
        SJMP LAB309
LABS15:
        CJNE A,#40H,LABS134
        LCALL READVEL
        MOV A,1EH
        CJNE A,#THREVEL1,LABS151
LABS152:
        MOV A,#090H  ; Turn key ON command
        LCALL TX
        MOV 83H,#41H
        MOV 82H,1DH
        MOVX A,@DPTR   ;Key No.
        LCALL TX
        NOP
        NOP
        NOP
        NOP
        NOP
        MOV A,1EH    ; Velocity  for  KEY ON
        LCALL TX
;
        MOV 83H,#43H
        MOV 82H,1DH
        MOV A,#S_4            ;Status  S 5 -> S 4
        MOVX @DPTR,A          ;Turn Key ON (At least 50 ms)
        MOV 83H,#42H
        MOV 82H,1DH
        MOV A,18H
        MOVX @DPTR,A           ;Current time -> 42xxH
        LJMP LAB306
LABS151:
        JNC LABS152
```

```
        SJMP LABS155

LABS134:
        LCALL READVEL
        MOV A,1EH
        CJNE A,#THREVEL2,LABS3
TUOFF:
        MOV 83H,#41H
        MOV 82H,1DH
        MOVX A,@DPTR           ; Key no.
        LCALL TX               ;Sent out key no.
        MOV A,#0H
        LCALL TX               ;Send out 0 velocity
;
        MOV 83H,#43H
        MOV 82H,1DH
        MOV A,#S_2            ;Status  S 3 or S 4 -> S 2
        MOVX @DPTR,A          ;Turn Key OFF (At least 30 ms)
LABS6:
        MOV 83H,#42H
        MOV 82H,1DH
        MOV A,18H
        MOVX @DPTR,A           ;Current time -> 42xxH
        LJMP LAB306
LABS3:
        JC TUOFF              ;Velocity < threshold jmp to TUOFF
        MOV 83H,#43H
        MOV 82H,1DH
        MOV A,#20H            ;Status  S 3 or S 4 -> S 3
        MOVX @DPTR,A
        SJMP LABS6
LAB304:
        LJMP LAB307
;
;
LAB309:
        ORL 0A8H,#84H          ;Enable INT1
        DEC 20H                ;L = L - 1
        MOV A,20H
        JNZ LAB304             ;If L <> 0 jmp to LAB307
        MOV DPTR,#4103H
        MOVX A,@DPTR
        JZ LAB320              ;If V = 0 jmp to LAB320
;-----------------------------------------------------------
        MOV 1FH,A              ;
```

```
            MOV DPTR,#4102H         ;
            MOV A,#0H               ;  10 T = 0
            MOVX @DPTR,A            ;  20 POP GSPE
    LAB308: POP 1DH                 ;  30 M(T) = GSPE
            ADD A,#010H             ;  40 T = T + 1
            MOV 82H,A               ;  50 IF T <> V THEN GOTO 20
            MOV A,1DH               ;
            MOVX @DPTR,A            ;
            MOV DPTR,#4102H         ;
            MOVX A,@DPTR            ;
            INC A                   ;
            MOVX @DPTR,A            ;
            CJNE A,1FH,LAB308       ;
    ;---------------------------------------------------------
    LAB320: LJMP LAB301
    ;
    LAB310: JC LAB311
            LJMP LAB312
    LAB331:JNC LAB322
            LJMP LAB321
    ;--------------------------------------
    ;   C <> 0
    ;--------------------------------------
    LAB302: ANL 0A8H,#0FBH      ;Disable INT1
            MOV DPTR,#4103H
            MOVX A,@DPTR
            MOV 1FH,A                   ;Get V
            MOV DPTR,#4100H
            MOVX A,@DPTR
            MOV 1DH,A
            CJNE A,#08,LAB310           ;If C > 8 jmp to LAB312
    ;
    ;   C <= 8
    ;
            LAB311: MOV DPTR,#4103H
            MOVX A,@DPTR
            ADD A,1DH                   ; C + V -> Accumulator
            CJNE A,#9H,LAB331           ;If C + V < 9 then jmp to LAB321
    ;---------------------------------------------------------------
    ;   C + V >= 9  --> Turn off key M(8 - C) .... M(V - 1)
    ;
    LAB322: MOV A,#8H
            CLR C
            SUBB A,1DH
            MOV 1EH,A
```

```
        MOV 1FH,A              ;V = 8 - C
LAB323: MOV A,1EH
        ADD A,#10H
        MOV 83H,#41H
        MOV 82H,A
        MOVX A,@DPTR
        MOV 1BH,A
        MOV 83H,#41H
        MOV 82H,1BH    ; Key no.
        MOVX A,@DPTR
        LCALL TX       ;Send key NO.
        MOV A,#0H      ;Send 0 velocity
        LCALL TX
        MOV 82H,1BH
        MOV 83H,#43H
        MOV A,#S_2
        MOVX @DPTR,A   ;Status -> S 1 (Turn OFF at least 50 ms)
        MOV 83H,#42H
        MOV 82H,1BH
        MOV A,18H
        MOVX @DPTR,A          ;Current time -> 42xxH
        INC 1EH
        MOV DPTR,#4103H
        MOVX A,@DPTR
        CJNE A,1EH,LAB323
LAB321: MOV DPTR,#4103H
        MOV A,1FH
        MOVX @DPTR,A            ;Store V
        JZ LAB332
;---------------------------------------------------
;
;  M() shift by C place
;
        MOV DPTR,#4117H
LAB313: MOVX A,@DPTR
        MOV 1EH,A             ;
        MOV A,82H             ;
        MOV 1FH,A             ;
        ADD A,1DH             ;
        MOV 82H,A             ;M(0)...M(7)  -->  M(C)...M(7+C)
        MOV A,1EH             ;
        MOVX @DPTR,A          ;
        MOV A,1FH             ;
        DEC A                 ;
        MOV 82H,A             ;
```

```
        CJNE A,#0FH,LAB313
;-----------------------------------------------------------
LAB332: MOV 1EH,1DH
        DEC 1EH
LAB314: MOV A,#1FH     ;
        ADD A,1DH              ;
        CLR C                  ;
        SUBB A,1EH             ;  10 FOR J = 0 TO C-1
        MOV 82H,A              ;  20 M(J) = IO(C-J-1)
        MOVX A,@DPTR           ;  30 NEXT J
        MOV 1FH,A              ;
        MOV A,#10H             ;
        ADD A,1EH              ;
        MOV 82H,A              ;
        MOV A,1FH              ;
        MOVX @DPTR,A           ;
        DEC 1EH                ;
        MOV A,1EH              ;
        CJNE A,#0FFH,LAB314
;-----------------------------------------------------------
        MOV DPTR,#4103H
        MOVX A,@DPTR           ;
        ADD A,1DH              ; V = C + V
        CJNE A,#9H,LAB325      ;
LAB326: MOV A,#8H      ;
LAB327: MOVX @DPTR,A
;-----------------------------------------------------------
        LJMP LAB315
LAB325:JNC LAB326
        SJMP LAB327
;-----------------------------------------------------------
;
;      C > 8 --> Turn off all keys in M() (from M(0)...M(V-1))
;
LAB312: MOV 1EH,#0H
LAB328: MOV A,1EH
        ADD A,#10H
        MOV 83H,#41H
        MOV 82H,A
        MOVX A,@DPTR
        MOV 1BH,A
        MOV 83H,#41H
        MOV 82H,1BH    ; Key no.
        MOVX A,@DPTR
        LCALL TX       ;Send key NO.
```

```
        MOV A,#0H       ;Send 0 velocity
        LCALL TX
        MOV 82H,1BH
        MOV 83H,#43H
        MOV A,#S_2
        MOVX @DPTR,A    ;Status -> S 1 (Turn OFF at least 50 ms )
        MOV 83H,#42H
        MOV 82H,1BH
        MOV A,18H
        MOVX @DPTR,A           ;Current time -> 42xxH
        INC 1EH
        MOV DPTR,#4103H
        MOVX A,@DPTR
        CJNE A,1EH,LAB328
;---------------------------------------------------
;
; MOV IO((I-1)...0) --> M(0...(I-1))
;
LAB330: MOV DPTR,#4101H
        MOVX A,@DPTR
        MOV 1DH,A
        MOV 1EH,A
LAB316: MOV A,#1FH
        ADD A,1EH
        MOV 82H,A
        MOVX A,@DPTR
        MOV 1FH,A
        MOV A,#11H
        ADD A,1DH
        CLR C
        SUBB A,1EH
        MOV 82H,A
        MOV A,1FH
        MOVX @DPTR,A
        DEC 1EH
        MOV A,1EH
        JNZ LAB316
;----------------------------------------------
        MOV A,1DH
        CJNE A,#8H,LAB317   ;If I <> 8 jmp to LAB317
;
LAB319: MOV A,#8
        MOV DPTR,#4103H
        MOVX @DPTR,A        ;  V = 8
;
```

```
LAB315:MOV DPTR,#4101H
        MOV A,#0H
        MOVX @DPTR,A       ;   I = 0
        DEC 82H
        MOVX @DPTR,A       ;   C = 0
        ORL 0A8H,#84H      ;Enable INT1
        LJMP LAB300
;------------------------------------------------------
;
; MOVE IO(7...I) TO M(I...7)
;
LAB317:MOV DPTR,#4101H
        MOVX A,@DPTR            ;Get I
        MOV 1DH,A
        MOV 1EH,A
LAB318: MOV A,#20H
        ADD A,1EH
        MOV 82H,A
        MOVX A,@DPTR
        MOV 1FH,A
        MOV A,#17H
        ADD A,1DH
        CLR C
        SUBB A,1EH
        MOV 82H,A
        MOV A,1FH
        MOVX @DPTR,A
        INC 1EH
        MOV A,1EH
        CJNE A,#8H,LAB318
        SJMP LAB319

;

       ;--------------------------------------------------;
       ;           READ VELOCITY SUBROUTINE               ;
       ;--------------------------------------------------;

READVEL:
        MOV A,1DH
        ADD A,#8H
        MOV 90H,A              ;Out GSPE+8 to port 1
        MOV A,#80H
LAB342: INC A
        JNZ LAB342             ;Delay 200 us
```

```
            MOV A,#0
            MOV DPTR,#0F001H
            MOVX @DPTR,A            ;Reset A/D converter
            NOP
            NOP
            NOP
            NOP
            MOV DPTR,#0F001H
            MOVX A,@DPTR            ;Read A/D converter
            ANL A,#07FH             ;MASK MSB
            MOV 1EH,A               ;VELOCITY
            RET
    ;
            ;------------------------------------------------;
            ;     CLEAR NOTE STATUS AREA (CLEAR STATUS 1)    ;
            ;------------------------------------------------;
    ;
    C_STATUS:
            MOV 1DH,#30H             ;Start from #38H
    LAB400:
            MOV 83H,#43H
            MOV 82H,1DH
            MOVX A,@DPTR             ;GET STATUS BYTE
            ANL A,#0F0H
            CJNE A,#10H,LAB401
            MOV 83H,#42H
            MOV 82H,1DH
            MOVX A,@DPTR             ;GET SET TIME
            LCALL CHKTM              ;CHECK IF 10 MS OUT
            JC LAB401                ;IF CARRY SET -> NOT YET
            MOV 83H,#43H
            MOV 82H,1DH
            MOVX A,@DPTR             ;GET STATUS BYTE
            ANL A,#0FH
            JZ LAB402
            MOVX A,@DPTR             ;GET STATUS BYTE
            DEC  A
            MOVX @DPTR,A
            MOV 83H,#42H
            MOV 82H,1DH
            MOV A,18H
            MOVX @DPTR,A            ;Current time -> 42xxH
            SJMP LAB401
    LAB402:
            MOV 83H,#43H
```

```
        MOV 82H,1DH
        MOV A,#0                ;STATUS S 2 -> S 1
        MOVX @DPTR,A
LAB401:
        INC 1DH                 ;Point to next channel
        MOV A,1DH
        ANL A,#0FH
        XRL A,#08H
        JNZ LAB400
        MOV A,1DH
        ADD A,#8H             ;Point to next key card
        MOV 1DH,A
        CJNE A,#80H,LAB400
        RET
;
;-----------------------------------------------------------------;
;          TIMER0 INTERRUPT ROUTINE    ( 1 MS/INT )                ;
;-----------------------------------------------------------------;
TMINT:
        INC 18H               ;Increase 1 ms timer
        MOV 8AH,#06CH    ;Set 1 ms timer0 low byte FOR XTAL = 11 MHZ
        MOV 8CH,#0FCH         ;Set 1 ms timer0 high byte
        POP 0D0H              ;Pop PSW
        RETI                  ;Return from TM0 INT
;-----------------------------------------------------------------;
;          TX SUBROUTINE : TRANSMIT (A) TO SERIAL PORT             ;
;-----------------------------------------------------------------;
TX:     ANL 98H,#0FDH    ;Clear tx flag
        MOV 99H,A
LAB3:   MOV A,98H        ;If data hasn't been tx
        ANL A,#02H       ; loop here
        JZ  LAB3
        RET
;
;-----------------------------------------------------------------;
;          TIMER CHECK SUBROUTINE : CHECK IF 10 MS OUT             ;
;    (BEFORE ENTER A CONTAINS THE TIME TO BE CHECKED)              ;
;    IF NOT YET                                                    ;
;       THEN SET CARRY FLAG                                        ;
;       ELSE CLEAR CARRY FLAG                                      ;
;-----------------------------------------------------------------;
CHKTM:  CLR C
        SUBB A,018H             ;current 1 ms timer
        XRL A,#0FFH
        INC A
```

```
            CJNE A,#0AH,LAB9       ;10*1 MS
LAB9:       RET
;
;
;
;----------------------------------------------------------------------;
;             INT1 INTERRUPT ROUTINE                                   ;
;----------------------------------------------------------------------;
INT_1:
            PUSH 0E0H              ;Push A
            PUSH 82H               ;Push DPTR-L
            PUSH 83H               ;Push DPTR-H
;
LAB001:     MOV  DPTR,#0F000H
            MOVX A,@DPTR
            MOV  1BH,A      ;Read GSPE (group selected priority encoder)
            PUSH 1BH              ;PUSH GSPE
            ADD  A,#08
            MOV 90H,A             ;Out to port 1
;
            MOV 83H,#43H
            MOV 82H,1BH
            MOVX A,@DPTR           ;Get this Note status byte
            JNZ LABI2              ;If ACTIVE jmp to LABI2

      ;-----------------------------------------;
      ;             This Note is OFF right now  ;
      ;-----------------------------------------;
LABI1:
            LCALL REVEL           ;READ VELOCITY
            LCALL K_ARRAY
            SJMP  LABA1

LABI2:      ANL A,#0F0H           ;MASK STATUS TIME COUNT NIBBLE
            CJNE A,#10H,LABI3
            MOV 83H,#43H
            MOV 82H,1BH
            ANL A,#0FH             ;MASK STATUS NIBBLE
            ORL A,#40H             ;Check Velocity 50 ms later
LABI3A:
            MOVX @DPTR,A           ;Status  -> S 5
            MOV 83H,#42H
            MOV 82H,1BH
            MOV A,18H
            MOVX @DPTR,A            ;Current time -> 42xxH
```

```
        SJMP LABA1
LABI3:  CJNE A,#20H,LABA1
        MOV 83H,#41H
        MOV 82H,1BH   ; Key no.
        MOVX A,@DPTR
        LCALL TX      ;Send key NO.
        MOV A,#0H     ;Send 0 velocity TURN NOTE OFF
        LCALL TX
        MOV 83H,#43H
        MOV 82H,1BH
        MOV A,#S_5          ;Check Velocity 30 ms later
        SJMP LABI3A
;
LABA1:
        POP 1BH          ;POP GSPE
LABA11:
        MOV DPTR,#0F002H ;Point to reset address
        MOV A,#0FFH
LAB021: MOVX @DPTR,A     ;RESET
        INC A
        JNZ LAB021
        MOV DPTR,#0F000H
        MOVX A,@DPTR
        CJNE  A,1BH,LABIOUT
        SJMP LABA11
LABIOUT:
        POP 83H                ;Pop DPTR-H
        POP 82H                ;Pop DPTR-L
        POP 0E0H               ;Pop A
        POP 0D0H               ;Pop PSW
        RETI                   ;Return from INT1
       ;---------------------------------------------------;
       ;          READ VELOCITY                            ;
       ;        IF VELOCITY > THRESHOLD1                   ;
       ;           THEN                                    ;
       ;             TURN KEY "ON" RIGHT NOW               ;
       ;           ELSE                                    ;
       ;             CHECK VELOCITY 20MS LATER             ;
       ;---------------------------------------------------;
REVEL:
        MOV DPTR,#0F001H
        MOVX @DPTR,A          ;Reset A/D converter
```

```
        NOP
        NOP
        NOP
        NOP
        MOV DPTR,#0F001H
        MOVX A,@DPTR        ;Read A/D converter
        ANL A,#07FH         ;Mask MSB
        MOV 1CH,A           ;VELOCITY
        CJNE A,#THREVEL1,LABR2
LABR1:
        MOV 83H,#43H
        MOV 82H,1BH
        MOV A,#42H          ;Check Velocity 20 ms later
        MOVX @DPTR,A        ;Status  -> S 5
LABR3:
        MOV A,#0
        MOV DPTR,#0F002H
        MOVX @DPTR,A        ;Reset this channel
        MOV 83H,#42H
        MOV 82H,1BH
        MOV A,18H
        MOVX @DPTR,A         ;Current time -> 42xxH
        RET
LABR2:
        JC LABR1
        MOV 83H,#43H
        MOV 82H,1BH         ;Change status byte
        MOV A,#S_4          ; Turn On at least 50 ms
        MOVX @DPTR,A        ; Status  -> S 4

        MOV A,#090H  ; Turn key ON command
        LCALL TX
        MOV 83H,#41H
        MOV 82H,1BH
        MOVX A,@DPTR   ;Key No.
        LCALL TX
        NOP
        NOP
        NOP
        NOP
        NOP
        MOV A,1CH   ; Velocity
        LCALL TX

        SJMP LABR3
```

```
;
        ;-----------------------------------------------------;
        ;           NOTE ARRAY HANDLE ROUTINE                  ;
        ; ALLOW ONLY 8 NOTES "ON" AT THE SAME TIME             ;
        ;-----------------------------------------------------;
;
;
K_ARRAY:
        MOV DPTR,#4100H
        MOVX A,@DPTR   ;Get C -> new data counter
        INC A
        MOVX @DPTR,A   ; C = C + 1

        CJNE A,#9H,LAB012     ;Check if C > 8
;
;  C >= 8 -> turn the oldest key OFF IO(I)
;
LAB013: PUSH 1BH        ;PUSH current channel no.
        MOV 82H,#01H
        MOVX A,@DPTR
        ADD A,#20H
        MOV 82H,A
        MOVX A,@DPTR   ;Get IO(I)
        MOV 1BH,A
        MOV 83H,#41H
        MOV 82H,1BH    ; Key no.
        MOVX A,@DPTR
        LCALL TX       ;Send key NO.
        MOV A,#0H      ;Send 0 velocity
        LCALL TX
        MOV 82H,1BH
        MOV 83H,#43H
        MOV A,#S_2
        MOVX @DPTR,A   ;Status -> S 2 (Turn OFF at least 30ms )
        MOV 83H,#42H
        MOV 82H,1BH
        MOV A,18H
        MOVX @DPTR,A          ;Current time -> 42xxH
        POP 1BH        ;POP current channel no.
LAB015:
        MOV DPTR,#4101H
        MOVX A,@DPTR   ;Get I -> IO() array pointer
        ADD A,#20H
        MOV 82H,A
        MOV A,1BH      ;Get channel NO.
```

```
        MOVX @DPTR,A  ;Store to IO()
        MOV 82H,#01H
        MOVX A,@DPTR
        INC A
        ANL A,#07H
        MOVX @DPTR,A  ; I = I + 1
        RET
;
LAB012: JC LAB015          ;C <=8
        SJMP LAB013        ;C > 8
;
END
```

```
     baud_counter = FFF5       27    191    192
         c_status = 2403      648    290
            chktm = 246B      717    308    659
        code_area = 2000        4     85
          initial = 2090       88     89
            int_1 = 2475      729    133    136    158    161
          k_array = 253E      863    751
           lab001 = 247B      734
           lab012 = 258B      909    869
           lab013 = 2547      873    910
           lab015 = 2577      895    909
           lab021 = 24D6      785    787
           lab201 = 2135      219    225
           lab202 = 212F      217    229    233
           lab203 = 209F      108    112
           lab204 = 2137      220    222
           lab207 = 216D      253    242
           lab208 = 2152      238    251
           lab209 = 2167      248    250
             lab3 = 2464      705    707
           lab300 = 219A      287    270    592
           lab301 = 217A      268    291    420
           lab302 = 229A      429    271
           lab304 = 2261      392    400
           lab305 = 21A5      295    289
           lab306 = 2195      283    309    358    384
           lab307 = 21AA      299    393
           lab308 = 227B      409    418
           lab309 = 2264      396    284    329
           lab310 = 2290      422    436
           lab311 = 22AC      440    422
           lab312 = 2349      530    423
           lab313 = 22FF      485    496
           lab314 = 231A      500    514
           lab315 = 23B0      586    523
           lab316 = 238A      563    577
           lab317 = 23BF      597    580
           lab318 = 23C7      601    615
           lab319 = 23AA      582    616
           lab320 = 228D      420    403
           lab321 = 22F4      476    425
           lab322 = 22B5      447    424
           lab323 = 22BE      452    475
           lab325 = 2345      524    519
           lab326 = 233F      520    524
           lab327 = 2341      521    525
```

```
          lab328 = 234C      531   554
          lab330 = 2382      559
          lab331 = 2295      424   443
          lab332 = 2315      498   479
          lab342 = 23ED      629   630
          lab400 = 2406      650   684   688
          lab401 = 2440      679   655   660   673
          lab402 = 2437      674   665
          lab410 = 20F0      173   181   185
            lab9 = 2474      722   721
           laba1 = 24CF      780   752   766   768
          laba11 = 24D1      782   791
           labi1 = 2490      749
           labi2 = 2498      754   744
           labi3 = 24B3      768   755
          labi3a = 24A7      760   778
         labiout = 24E3      792   790
           labr1 = 24FF      820   835
           labr2 = 2518      834   819
           labr3 = 2508      825   855
         labs134 = 2228      363   331
          labs15 = 21EB      330   319
         labs151 = 2224      359   334
         labs152 = 21F6      335   360
         labs155 = 21DA      320   361
          labs20 = 2183      272   315
           labs3 = 2254      385   366
           labs6 = 2248      379   391
    main_program = 2177      265
       ram_error = 20E8      163   145   148   151   155   158   161   164
         readvel = 23E5      624   332   364
          reset1 = 2093       98   100
           revel = 24EC      808   750
             s_2 = 0013       31   377   466   545   888
             s_4 = 0035       32   352   838
             s_5 = 0043       33   777
        settable = 20EA      170   162
        threvel1 = 0004       24   334   819
        threvel2 = 0001       25   366
        tim0_c_h = 00FC       29
        tim0_c_l = 006C       28
           tmint = 2454      694   123   126   148   151
           tuoff = 2230      367   386
              tx = 245F      703   337   341   348   371   373   461
                             463   540   542   772   774   842   846
                             853   883   885
```

PROGRAM : MIDI DATA COLLECTION BASIC PROGRAM

```
10      XTAL=4000000
20      IF XBY(5000H) <> 0 THEN 100
30      CALL 4100H  :  REM INITIALIZATION
40      RCAP2=65532 :  REM FOR 31250 BAUD
50      CALL 4200H  :  REM COLLECT MIDI DATA
100     CALL 4100H  :  REM INITIALIZATION
110     RCAP2=65532 :  REM FOR 312500 BAUD
120     PORT1=PORT1.AND.0FEH : REM ENABLE MIDI OUT
130     CALL 4300H  :  REM OUTPUT MIDI DATA
140     PORT1=PORT1.OR.001H  : REM DISABLE MIDI OUT
150     RCAP2=65523 :  REM FOR 9600 BAUD
160     IE=0 : REM DISABLE ALL INTERRUPT
170     STOP
```

NOTE: FOLLOWING PAGES ARE ASSEMBLY PROGRAM CALLED BY THIS BASIC PROGRAM.

```
;------------------------------------------------------------------
;
;------------------------------------------------------------------
;------------------------------------------------------------------
;          TIMER0 INTERRUPT VECTOR
;------------------------------------------------------------------
ORG 400BH
         LJMP TMINT
;------------------------------------------------------------------
;          INITIALIZATION CALLED BY MAIN PROGRAM
;------------------------------------------------------------------
ORG 4100H
         ANL 88H,#0CFH        ;Disable timer0
         MOV 18H,#0H          ;Reset timer MSB (65536 ms/count)
         MOV 19H,#0H          ;Reset timer     (256 ms/count)
         MOV 1AH,#0H          ;Reset timer LSB (1 ms/count)
         MOV 8AH,#02FH        ;Set timer0 low byte(1 MS TIMER)
         MOV 8CH,#0F6H        ;Set timer0 high byte(1 MS TIMER)
         ANL 89H,#0F1H        ;Set timer0 to 16 bit timer
         ORL 88H,#30H         ;Enable timer0
         ORL 0A8H,#82H        ;Enable timer0 interrupt
         RET
;------------------------------------------------------------------
;          TIMER0 INTERRUPT ROUTINE   ( 1 MS/INT )
;------------------------------------------------------------------
;
TMINT:
         PUSH 0E0H            ;PUSH A
         INC 1AH              ;Increase 1 ms timer low byte
         MOV A,1AH
         JNZ TMLAB1
         INC 19H              ;Increase 1 ms timer middle byte
         MOV A,19H
         JNZ TMLAB1
         INC 18H              ;Increase 1 ms timer high byte
TMLAB1:
         MOV 8AH,#02FH        ;Set timer0 low byte(1 ms at XTAL = 4MH
         MOV 8CH,#0F6H       ;Set timer0 high byte(1 ms at XTAL = 4MH
         POP 0E0H             ;POP A
         POP 0D0H             ;POP PSW
         RETI
;
;------------------------------------------------------------------
;          CATCH DATA PROGRAM
;------------------------------------------------------------------
```

```
ORG 4200H
         PUSH 0E0H            ;Push A
         PUSH 82H             ;Push DPTR-L
         PUSH 83H             ;Push DPTR-H
         ANL 98H,#0FEH        ;Clear SCON.0 (receives data flag)
         MOV DPTR,#5000H      ;Reset data pointer
LAB2:    JBC SCON.0,LAB1      ;If received data jmp to LAB1
         JB P1.7,LAB2         ;If SW1 is not pushed jmp to LAB2
LAB3:    PUSH 82H             ;
         MOV A,83H            ;
         MOV DPTR,#4280H      ;
         MOVX @DPTR,A         ;Store DPTR-H to #4280H
         INC DPTR             ;
         POP 0E0H             ;
         MOVX @DPTR,A         ;Store DPTR-L to #4281H
         POP 83H              ;POP DPTR-H
         POP 82H              ;POP DPTR-L
         POP 0E0H             ;POP A
         RET
;
LAB1:
         MOV A,SBUF           ;Get data from serial buffer
         MOVX @DPTR,A         ;Store it to data area
         INC DPTR             ;Data pointer increases 1
         ANL 0A8H,#07FH       ;Disable timer0 INT
         MOV A,18H            ;Get 1 ms timer low byte
         MOVX @DPTR,A         ;Store it
         INC DPTR             ;Data pointer increases 1
         MOV A,19H            ;Get 1 ms timer middle byte
         MOVX @DPTR,A         ;Store it
         INC DPTR             ;Data pointer increases 1
         MOV A,1AH            ;Get 1 ms timer high byte
         MOVX @DPTR,A         ;Store it
         INC DPTR             ;Data pointer increases 1
         ORL 0A8H,#82H        ;Enable timer0 interrupt
         MOV A,83H            ;Get DPTR-H
         CJNE A,#60H,LAB2     ;If DPTR-H = #60H -> Data file full
         LJMP LAB3
;--------------------------------------------------------------------
;          SEND DATA PROGRAM
;--------------------------------------------------------------------
ORG 4300H
        PUSH 0E0H             ;Push A
        PUSH 82H              ;Push DPTR-L
        PUSH 83H              ;Push DPTR-H
```

```
           MOV DPTR,#5000H        ;Reset data pointer
    LAB4:
           MOVX A,@DPTR           ;Get MIDI data
           CJNE A,#0F8H,LAB5      ;If not system clock F8H jmp to LAB5
           INC DPTR               ;
           INC DPTR               ;
           INC DPTR               ; data pointer increases 4
           INC DPTR               ;
           SJMP LAB4
    LAB5:
           MOV 1BH,A              ;MIDI DATA( EXCEPT 0F8H )
           INC DPTR
           MOVX A,@DPTR
           MOV 1CH,A              ;HIGH BYTE OF 1 MS COUNT
           INC DPTR
           MOVX A,@DPTR
           MOV 1DH,A              ;MIDDLE BYTE OF 1 MS COUNT
           INC DPTR
           MOVX A,@DPTR
           MOV 1EH,A              ;LOW BYTE OF 1 MS COUNT
           INC DPTR
           MOV A,1BH              ;-------------------------
           JNZ LAB6               ;If both MIDI data & 3 bytes
           MOV A,1CH              ;    1 ms timer of this data
           JNZ LAB6               ;    all equal 0 then this is
           MOV A,1DH              ;    the end of data area
           JNZ LAB6               ;    JMP to LAB13.
           MOV A,1EH              ;Otherwise goto LAB6.
           JZ  LAB13              ;-------------------------
    LAB6:
           NOP
           ANL 0A8H,#07FH         ;Disable timer0 INT
           MOV A,18H              ;------------------------
           CJNE A,1CH,LAB8        ;Compare 1 ms timer of this
           MOV A,19H              ;  MIDI data with current 1 ms
           CJNE A,1DH,LAB8        ;  timer. If time has not passed
           MOV A,1AH              ;  jmp to LAB6.
           CJNE A,1EH,LAB8        ;------------------------
    LAB12:
           ORL 0A8H,#82H         ;Enable timer0 interrupt
           ANL 98H,#0FCH         ;
           MOV 99H,1BH           ;TX MIDI DATA
    LAB11: MOV A,98H             ;
           ANL A,#02H            ;Check if DATA has been sent out
           JZ  LAB11             ;If not wait here
```

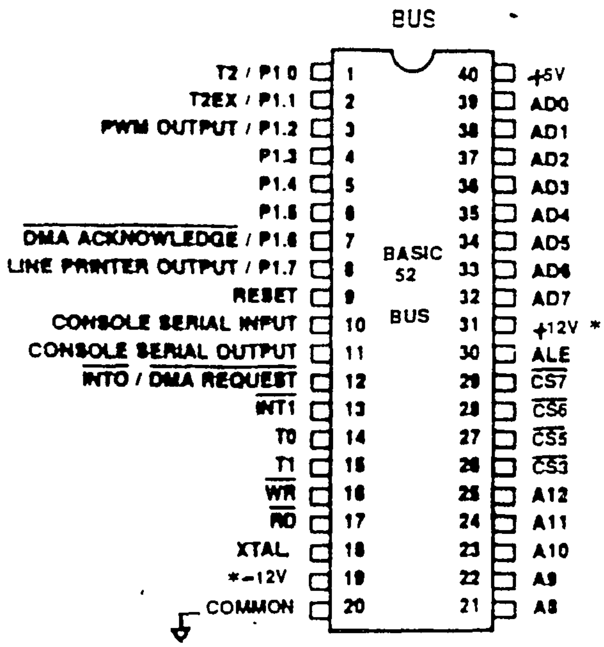
```
            MOV A,1BH           ;
            CJNE A,#0FCH,LAB4   ;If the DATA is #0FCH
    LAB13:                      ;   then return to main program
            POP 83H             ;POP DPTR-H
            POP 82H             ;POP DPTR-L
            POP 0E0H            ;POP A
            RET
    LAB8:
            JNC LAB12           ;IF NC THEN CURRENT TIME  > SET TIME
            ORL 0A8H,#82H       ;Enable timer0 interrupt
            SJMP LAB6
    ;;
    END
```

```
    lab1 = 4225       66     52
   lab11 = 434E      133    135
   lab12 = 4345      129    144
   lab13 = 4359      138    119
    lab2 = 420C       52     53     82
    lab3 = 4212       54     83
    lab4 = 4309       92     99    137
    lab5 = 4313      100     94
    lab6 = 4332      120    113    115    117    146
    lab8 = 4360      143    124    126    128
   tmint = 411C       27      8
  tmlab1 = 412C       36     31     34
```

APPENDIX C

ARCHITECTURE OF SBC

Canonical SBC Bus (low profile 40 pin header, .1X.1)



* Beware DO NOT let the ±12V rails CONTACT ANY of the other bus pins!

All the functions of the 8052 itself appear on the bus, which is pinned out to match the chip, except:

(1) B.31 is used for +12V. The corresponding pin on the microC has no use externally.

(2) B.18 is the buffered output of the Xtal, and B.19 is used for -12V.

(3) B.26 to B.29 carry four chip selects: Recall A13-15 is decoded, and PSEN is anded with RD above 04000H, so no chip functions are excluded from the bus by these assignments, and the chip selects are enormously useful in peripheral design.

Memory Map

The 16 bit memory address space is decoded on 8K boundaries. The architecture is Harvard up to 03FFFH and Von Neuman above. Sockets are provided as follows:

```
U7  (02000H-03FFFH) Code CS1\ - esp microMint

U8  (00000H-01FFFH) Data CS0\ - BASIC, obligatory.
                                01FFFH is MTOP.

U9  (04000H-05FFFH) D/C  CS2\ - Battery backed INT vectors,
                                general purpose.

U10 (08000H-09FFFH) D/C  CS4\ - BASIC stores BASIC programs
                                bottom up - top down
                                is general purpose.
                                Battery backed.
```

Other blocks are presented to the bus as follows:

```
CS7\ - B.29   (0E000H-0FFFFH)
CS6\ - B.28   (0C000H-0DFFFH)
CS5\ - B.27   (0A000H-0BFFFH)
CS4\ - B.26   (06000H-07FFFH)
```

Programming multiplexer on Port 1.3-.5

The signal, CS1\+WR\, sets up the program store mode. A RD there will exit, as will completion of programming task. XBY(9999) is an easily typed way to generate CS1\. DATA CS1\ is used only for this purpose.

Other detail

Battery backing is permanent, the potential being maintained by a Ni-Cad, trickle charged when the SBC is plug in. U9 and U10 are backed.

Serial connections (D-9F) are buffered by RS-232 inverting drivers/receivers. The RS-232 chip uses only 5V.

Power: 5V at 175ma, ±12 Volt rails are passed through to the bus for convenience, (+12 B.31;-12 B.19); they are not used by SBC itself.

A 4 MHz Xtal is used as a laboratory convenience. (11.0592 MHz is handy). You must alert Basic of this fact by running

XTAL = 4000000

Program Storage Mode

The SBC uses a battery backed RAM in U10 (08000H-09FFFH) in place of an EPROM. At the outset you must "erase" the memory block -- set all bits to one -- by RUNning, for example,

```
10 FOR I = 08000H TO 09FFFH
20 XBY(I) = 0FFH
30 NEXT
```

To use any or all of MCS52 Basic EPROM file commands you must first enter the SBC program storage mode by the following action:

```
XBY(9999) = 0  <CR>
```

The LED (next to reset button) will light.

Exit is automatic at the end of a programming cycle, or the command

```
X = XBY(9999)  <CR>
```

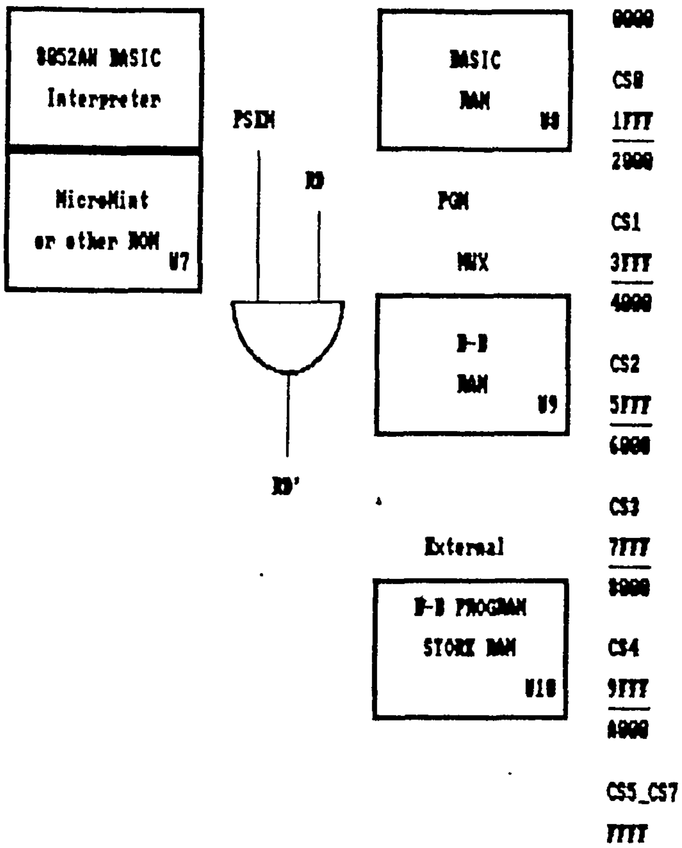
will get you out if you change your mind.

In the program storage mode, port 1 pins on the Bus, B.4, B.5, and B.6 are floating. Regard for this fact may be necessary in a peripheral design that employs P1.3-.5.

<u>N.B.:</u> Just before and during a program storage operation, P1.3 and .4 MUST BE HIGH, the reset condition. Be careful not to RUN a program which might take these pins low: Remember, once they are made low, they stay low even after program has run. For security you might wish to execute

PORT1=PORT1.OR.018H <CR>

before entering the program store mode, and thereby set P1.3 and .4 high for sure. If P1.3 (or possibly P1.4) is low when you enter program store mode. The AD bus demultiplexer, ALE, is disabled and the only cure is RESET, which will, of course, clear away the RAM program you may have been fond of.



SBC Memory Map

Fig.INT.2

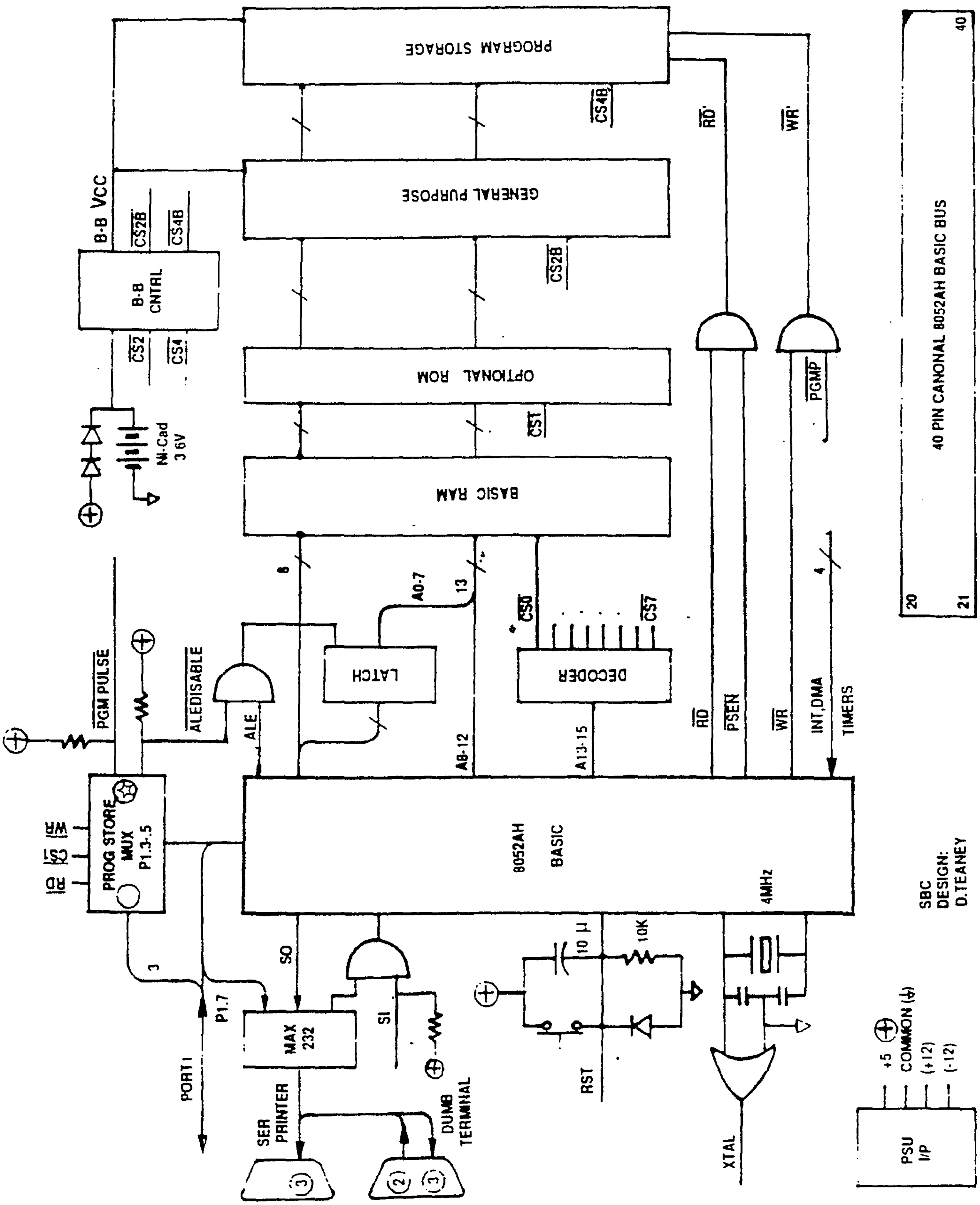
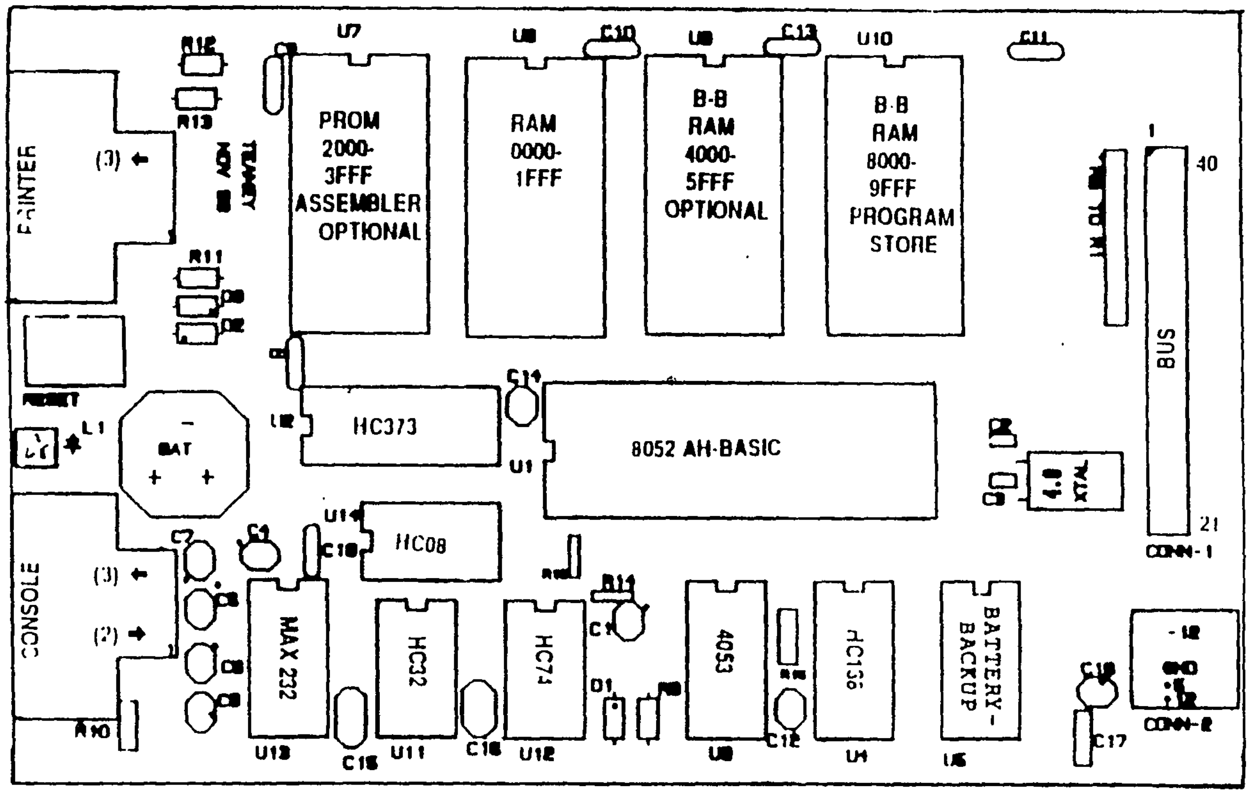
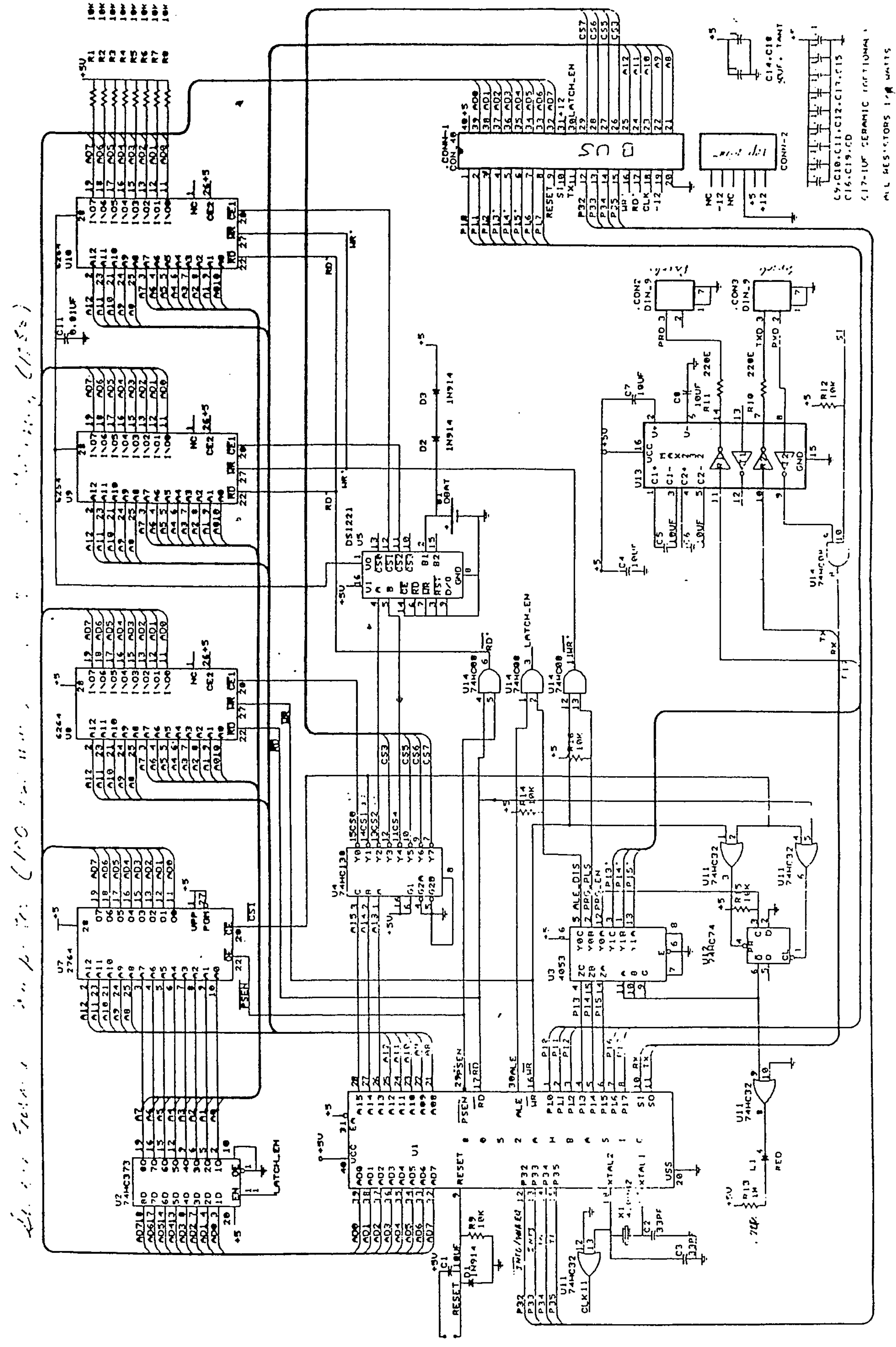

PROGRAM STORAGE
GENERAL PURPOSE
OPTIONAL ROM
BASIC RAM
LATCH
DECODER
8052AH
BASIC
4MHz
B-B CNTRL
B-B VCC
Ni-Cad
3.6V
PGM PULSE
ALE DISABLE
ALE
PROG STORE
MUX
P1.3-.5
MAX
232
SO
SI
P1.7
PORT1
SER
PRINTER
DUMB
TERMINAL
RST
XTAL
10K
A0-7
A8-12
A13-15
INT,DMA
TIMERS
PGMP
40 PIN CANONAL 8052AH BASIC BUS
PSU
I/P
+5
COMMON
(+12)
(-12)
SBC
DESIGN:
D.TEANEY

Fig.INT.3



Single Board Computer

Fig. INT. 4


U1
U2 74HC373
U7 2764
U8 6264
U9 6264
U10 6264
U4 74HC138
U5 DS1221
U3 4053
U13
U11 74HC32
U14 74HC08
CONN-1 CON_40
CONN-2
LATCH_EN
ALL RESISTORS 1/4 WATTS

REFERENCES

1. **INTERFACE TO 8052AH-BASIC MICROCONTROLLER FOR A MULTICHANNEL PRIORITY INTERRUPT SYSTEM,** by Chen-Tung Mo, NJIT EE department master thesis 1988.
2. **MULTICHANNEL PRIORITY INTERRUPT SYSTEM,** by Claudio Bernal, NJIT EE department master thesis 1988.
3. **MIDI 1.0 DETAILED SPECIFICATION**, The International MIDI Association, 1988.
4. **MCS BASIC-52 User manual**, Intel Corporation, 1986.
5. **Microcontroller Handbook**, Intel Corporation, 1986.
6. **BASIC-52 Extensions Manual ROM A+B**, Micromint Inc., 1986.